AD-A234 925

ITATION PAGE

Form Approved
OMB No. 0704-0188

ıd to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this rden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson fice of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	FINAL 19 Dec 88 to 18 Dec 90

4. TITLE AND SUBTITLE
GENERIC TASKS FOR KNOWLEDGE-BASED PROBLEM SOLVING: EXTENSION AND NEW DIRECTIONS

5. FUNDING NUMBERS
AFOSR-89-0250

61102F 2304/A7

6. AUTHOR(S)
B. Chandrasekaran

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
THE OHIO STATE UNIVERISTY
1314 KINNEAR ROAD
COLUMBUS, OH 43212-1994

8. PERFORMING ORGANIZATION REPORT NUMBER
AFOSR-TR- 91 0252

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFOSR/NM
Bldg 410
Bolling AFB DC 20332-6448

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFOSR-89-0250

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release;
distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

AFOSR-sponsored research completed during 1989 and 1990 has advanced the basic understanding of the structure of tasks in problem solving and how method selection can occur within knowledge based problem solving systems. The important notion of task-structure, providing a means of characterizing the relation of task, method, and subtask, aids in the construction of large, complex, and versatile systems. The analysis of the design task provides a good illustration of task structure methodology. Complexity analysis complement our understanding of the computational resource requirments of various cases of central solving tasks.

14. SUBJECT TERMS

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

THE OHIO STATE UNIVERSITY

Generic Tasks for Knowledge-Based Problem Solving: Extension and New Directions

B. Chandrasekaran
Department of Computer and Information Science

Air Force Office of Scientific Research
Bolling Air Force Base, D.C. 20332-6448

Grant No. AFOSR-89-0250
Final Report
RF Project No. 767305/721852

February 1991

A-1

Final Report under AFOSR Grant 89–0250 Generic Tasks for Knowledge-Based Problem Solving: Extensions and New Directions, December 19, 1988 to December 18, 1990

B. Chandrasekaran, Principal Investigator

February 27, 1991

Introduction

The Laboratory for Artificial Intelligence Research (LAIR) at The Ohio State University held AFOSR grant number 89-0250 (which succeeded a series of earlier grants, the most recent being 87-0090). LAIR's research program has focused on knowledge based systems, and AFOSR support has been central in supporting basic scientific research on problem solving tasks and on organizing and indexing systems for the knowledge utilized by problem solving tasks. LAIR's research contributions to understanding knowledge-based reasoning variety of advances and accomplishments. Perhaps the best known accomplishment has been the development of the generic task approach itself and its development into the broader research program exploring task specific architectures to which a variety of prominent researchers, such as W. Clancey, J. McDermott and L. Steels, also contribute[13, 15, 18, 8].

LAIR research on knowledge-based problem-solving began by delineating **generic information processing tasks**. A generic task is a primitive type of knowledge-based reasoning. Each generic task is functionally specified by its input and output, and is characterized by the organization of knowledge and control of problem-solving appropriate for the task. An important consequence of identifying generic tasks is that they provide a framework to characterize knowledge-based reasoning. Also, generic tasks provide high-level building blocks for expert system design and prototyping. If a complex real-world task can be decomposed into several generic tasks, and if we know how to perform each of these tasks, then there is a basis for concluding that the complex task can be successfully performed by an integrated system.

LAIR has also developed a functional representation language for one organizational scheme of our knowledge about the structure and behavior of devices [17]. The basic idea is that aspects of an agent's model of how a device works can be represented in a way that shows how an intended *function* is accomplished as a result of the *behaviors* of its components,

leading tc a series of *states* of the device. The Functional Representation Language has more recently been used to represent how aspects of *plans* and *programs*, with both viewed as abstract devices. FR representations are being applied to issues of organizing complex and ultimately large technical knowledge bases for multifunctional problem solving systems that can support design, diagnosis, prediction, explanation, and which can also contribute to tutorial systems.

In our 1989 report on basic research results from the LAIR, we summarized a variety of results that we have obtained in understanding knowledge based system construction and architecture. We summarized three generic task integration that had been pursued in LAIR's Generic Task Toolset [11], Bill Punch's TIPS system [16], and the ongoing work on abduction in SOAR [10]. We also reported on the the development of task-oriented methodology [4], and the general pattern of this method has been applied to the design task in a paper included with this report. [5] Our 1989 report discussed the systems of Tanner [19] and Keuneke [12] that provided explanatory capability for knowledge based systems. Finally, some work of Goel [9] on functional representation of device structure and behavior and the development of indexing schemes for redesign tasks was reviewed.

In this final report, we will report on continuations of previous research and also on some new directions that we have explored as part of our AFOSR grant.

Extensions

In the present section, recent research results that elaborate and develop previously identified issues and projects will be described. The major results arise from projects that are continuations of the following areas of research: theory building for task structures, novel architectures for integrating problem solving tasks, extensions of the range of domains to which LAIR task architectures and knowledge representation structures have been applied, improvements in the performance features of knowledge based systems (robustness, correctness, and efficiency). Also some applied theoretical results on the computational complexity of variants of two important problem solving tasks (abductive assembly and planning) will be described.

1. A task structure for design has been proposed based upon an analysis of a general class of methods called "Propose-Critique-Modify" methods. The task structure is constructed by identifying a range of methods for each task. For each method, the knowledge needed and the subtasks that it sets up are identified. This recursive style of analysis provides a framework in which we can understand a number of particular proposals for design problem solving as specific combinations of tasks, methods and subtasks. Most of the subtasks are not really specific to design as such. The analysis shows that there is no one ideal method for design, and good design problem solving is a result of recursively selecting methods based on a number of criteria including knowledge availability. How the task analysis can help in knowledge acquisition and system design is discussed. A copy of the paper is included with this report [5].

2. Functional representations of program specifications have been combined with grammatical and axiomatic semantic knowledge about program implementations to yield problem solving systems that can test implementations and arrive at justifications for program implementation correctness or at diagnoses of error in implementations. [1] This work uses the representation of device understanding reported in Sembugamoorthy and Chandrasekaran [17] that established the groundwork for representing understanding of a device by specifying the roles that components of a device play in its behavior. For non-trivial physical devices, it is often difficult to find an appropriate formal language to describe states of the device. However, when thinking of a computer program as a device, though programs share modularity of design with physical devices, they also have a formal description of their behavior (specified by the formal semantics of the programming language) with which it is possible to justify conclusions about the programs. A certain form of mathematical induction can yield a correctness proof for the program. In addition to resolving some open problems concerning the consistency of representations and the semantics of states, it is shown how such a representation of understanding can be used to automatically verify and correct programs.

3. In building larger and more complex knowledge base systems, the need has arisen to be able to identify errors and correct them–a problem of knowledge-base refinement. The task of corrective learning is: given a problem solving system that has produced an answer unacceptably different from the correct answer, identify how the system made the error (using the difference between the correct and erroneous answers) and then correct the problem solving system so it does not happen again. Functional representations have been used in building a system capable of corrective learning. [20]

   The credit assignment needed for this corrective process is an explanation-based one; that is, alternative explanations for why the knowledge-based system erred are proposed and the best one is selected. The credit assignment system examines trace explanations detailing how a knowledge-based system derived its answer. The explanations examined by the system are of two types: First, the credit assignment system devises error hypotheses that explain the deviation between the erroneous and real answer, and these explanations are examined. Second, the causal stories about what has happened in the domain themselves provide explanations. The main thesis on credit assignment is that capturing problem solving primitives at an appropriate level of abstraction for the available explanations guides the search through the space of potential candidates effectively.

   The first step in this task is the identification the set of error candidates potentially causing the mistake. Using generic, task specific architectures for designing knowledge-based systems greatly simplifies the identification of this set. Each task is modeled as a device (using the Functional Representation), and this representation is used to map from steps in the problem solving behavior to knowledge identifying error candidates. The second step in placing credit for a mistake is to evaluate the proposed error candidates so that the best set of error candidates accounting for the knowledge-based

system error is selected. A form of model-based reasoning is used to map the knowledge used by the knowledge-based system to a model representing the "complete" domain theory. One approach for representing this knowledge is to define a functional model of the problem solver and the domain. By relating the knowledge-based system's knowledge to the domain model and then by reasoning with this model, the explanation of the knowledge-based system's answer can be compared with an expianation of the expert's answer to yield locations for corrections in the system. This work differs from most work on this type of problem because the search for credit involves forming justified changes through the use of explanations rather than using statistical or heuristic changes; a paper is included with this report that explains how the error candidates are generated.

4. Computational complexity results have been obtained that allow fine discriminations to be made between tractable (polynomial) and nontractable (usually NP) cases of problem solving tasks of interest to knowledge based system builders. Abduction construed as a set covering problem (where hypotheses cover data sets) has been thoroughly characterized. The final paper for these results is included in this report.[3] Results have recently been obtained for planning problems that involve the selection of operators to get to some goal state from a given initial state, and a paper is included for this result also [2].

5. Soar architectural realizations for Generic Tasks have been constructed for hierarchical classification and for simple versions of abductive assembly. The theoretical basis for this mapping of SOAR constructs and generic tasks is discussed in a paper included with this report. [10] An initial large explanatory prototype testing the Soar implementation has been constructed

New Directions

1. The knowledge based system research tradition has generally made use of a variety of distinctions between "deeper" and "shallower" forms of knowledge that have been of some use in discussing comparative aspects of performance of systems accessing this knowledge. Unfortunately some of the terminology used in these discussions has been seen to be in need of some clarification. Some needed clarification of the concepts needed for discussion have been presented in a paper included with this report. [6]

2. Multifunctional knowledge based systems will integrate several reasoning capabilities, some in the service of others at various points in the problem solving process. Prediction and consequence finding, for example, can be important in testing whether a design should work, in checking whether a conjectured diagnostic scenario will work, and so on. Systems for carrying out these predictive subtasks should be able to integrate both quantitative and qualitative predictive tasks. In addition, the repertoire of qualitative predictive methods has not been fully elaborated. Visual spatial reasoning is one problem solving method that can accomplish some qualitative predictive tasks. [14]

3. Finally, both biological and cognitive psychological theorizing has been presumed to be relevant to the constructive and design portions of artificial intelligence research. Rational analysis (RA) has been proposed as a methodology that may illuminate how a mechanistic realization of mental capabilities may be found. I have argued that RA may be of help in drawing the distinction between the content that problem solving mechanisms operate upon, but is not necessarily useful in providing guidance about the abstract mechanisms of cognition. These arguments are developed in a paper included with this report [7].

Conclusion

AFOSR-sponsored research completed during 1989 and 1990 has advanced the basic understanding of the structure of tasks in problem solving and how method selection can occur within knowledge based problem solving systems. The important notion of task-structure, providing a means of characterizing the relation of task, method, and subtask, aids in the construction of large, complex, and versatile systems. The analysis of the design task provides a good illustration of task structure methodology. Complexity analyses complement our understanding of the computational resource requirements of various cases of central problem solving tasks.

Improved ways of understanding how knowledge can be used to guide method selection have become available and have been incorporated in several schemes for building integrated generic task systems for abduction in a universal subgoaling architecture (SOAR), program testing and corrective learning. During 1990, integration issues have been combined with issues about the organization, indexing, and access of knowledge of devices using the functional representation language. The scope of the FR language has been investigated in both novel domains and in novel uses (such as in the representation of a system's own task structure). In other research on multifunctional knowledge base systems, principles for handling the organization of complex device knowledge (including various class hierarchies) have been combined with the hierarchical organization of the original FR scheme.

References

[1] Dean Allemang. Understanding Programs as Devices. Dissertation, Advisor: B. Chandrasekaran, The Ohio State University, 1990.

[2] Tom Bylander. Complexity Results for Planning. *Artificial Intelligence*, 0:0, 1990.

[3] Tom Bylander. Computational Complexity of Abduction. *Artificial Intelligence*, 0:0, 1990.

[4] B. Chandrasekaran. Task-Structures, Knowledge Acquisition, and Learning. *International Journal of Machine Learning*, 4:339–345, 1989.

[5] B. Chandrasekaran. Design problem solving: A task analysis. *Artificial Intelligence Magazine*, 11(4):59–71, 1990.

[6] B. Chandrasekaran. Models vs Rules, Deep vs Compiled, Content vs Form: Some Distinctions in Knowledge Systems Research. *IEEE Expert*, To Appear(4):00, 1991.

[7] B. Chandrasekaran. Response to 'Is Human Cognition Adaptive' by J.R. Anderson *Brain and Behavior*, To Appear(00):00, 1991.

[8] W. J. Clancey. Heuristic classification. *Artificial Intelligence*, 27:289–350, 1985.

[9] Ashok Kumar Goel. Integration of Case-Based Reasoning for Adaptive Design Problem Solving. Dissertation, Advisor: B. Chandrasekaran, The Ohio State University, 1989.

[10] Todd R. Johnson, Jr. Jack W. Smith, and B. Chandrasekaran. Generic Tasks and Soar. Laboratory for AI Research Technical Report, 1989.

[11] John Josephson, Diana Smetters, and Others. The Generic Task Toolset, Fafner Release 1.0, Introduction. Laboratory for AI Research Technical Report, 1989.

[12] Anne M. Keuneke. Machine Understanding of Devices Causal Explanation of Diagnostic Conclusion. Dissertation, Advisor: B. Chandrasekaran, The Ohio State University, 1989.

[13] J. McDermott. A Taxonomy of Problem-Solving Methods. In *Automating Knowledge Acquisition for Expert Systems*, pages 225–256. Kluwer, Boston, 1988.

[14] Hari Narayanan and B. Chandrasekaran. Reasoning Visually about Spatial Interactions. LAIR Technical Report, 1991.

[15] Alan Newell. Reasoning, Problem Solving and Decision Process: The Problem Space as a Fundamental Category. In *Attention and Performance*, pages 693–718. Lawrence Erlbaum, 1980.

[16] William F. Punch. A Diagnosis System Using a Task Integrated Problem Solver Architecture (TIPS), Including Causal Reasoning. Dissertation, Advisor: B. Chandrasekaran, The Ohio State University, 1989.

[17] V. Sembugamoorthy and B. Chandrasekaran. Functional representation of devices and compilation of diagnostic problem solving systems. In C.K. Riesbeck J.L. Kolodner, editor, *Experience, Memory, and Reasoning*. Lawrence Erlbaum, 1986.

[18] L. Steels. Components of expertise. *AI Magazine*, 11:28–49, 1990.

[19] Michael C. Tanner. Explaining Knowledge Systems: Justifying Diagnostic Conclusions. Dissertation, Advisor: B. Chandrasekaran, The Ohio State University, 1989.

[20] Michael Weintraub and Tom Bylander. Corrective Learning by Generating and Comparing Explanations. In *AAAI Workshop on Explanation*, June 1990.

Selected Bibliography of AFOSR-supported OSU LAIR Research

References

[1] Dean Allemang. Understanding Programs as Devices. Dissertation, Advisor: B. Chandrasekaran, The Ohio State University, 1990.

[2] Dean Allemang, Michael C. Tanner, Tom Bylander, and John R. Josepshson. On the computational complexity of hypothesis assembly. In *Proceedings of IJCAI-87, Milan, Italy, August, 1987*. January 1987. Also available as OSU-LAIR Technical Report.

[3] D. Brown and B. Chandrasekaran. Design: An information processing level analysis. In *Design Problem Solving: Knowledge Structures and Control Strategies*. Pittman: Morgan-Kaufmann, 1989.

[4] D. C. Brown and B. Chandrasekaran. *Design Problem Solving: Knowledge Structures and Control Strategies*. Morgan Kaufmann, San Mateo, California, 1989.

[5] T. Bylander and B. Chandrasekaran. Generic tasks for knowledge-based reasoning: the "right" level of abstraction for knowledge acquisition. *International Journal of Man-Machine Studies*, 26:231–243, 1987.

[6] T. Bylander, B. Chandrasekaran, and J. Josephson. The generic task toolset. In *Proceedings of Second International Conference on Human-Computer Interaction*, Honolulu, Hawaii, August 1987.

[7] T. Bylander, J. Smith, and J. Svirbely. Qualitative representation of behavior in the medical domain. In *Proceedings of The Fifth Conference on Medical Informatics*, pages 7–11, Washington, D.C., October 26-30 1986. Conference on Medical Informatics.

[8] Tom Bylander. Primitives for reasoning about the behavior of devices. In *Proceedings of the AAAI Workshop on Qualitative Physics*, Urbana, Illinois, May 1987.

[9] Tom Bylander. Some causal models are deeper than others. In *Proceedings Ninth Annual Conference of the Cognitive Science Society*, Seattle, Washington, July 1987.

[10] Tom Bylander. Using consolidation for reasoning about devices. Technical report, The Ohio State University, 1987.

[11] Tom Bylander. A critique of qualitative simulation from a consolidation viewpoint. *IEEE Transactions on Systems, Man, and Cybernetics*, 18(2):252–268, March-April 1988.

[12] Tom Bylander and Michael A. Weintraub. A corrective learning procedure using different explanatory types. In *Proceedings of the AAAI Mini-Symposium on Explanation-Based Learning*, Stanford University, Palo Alto, California, March 1988.

[13] B. Chandrasekaran. Task-Structures, Knowledge Acquisition, and Learning. *International Journal of Machine Learning*, m:n, 1989.

[14] B. Chandrasekaran. Design problem solving: A task analysis. *Artificial Intelligence Magazine*, 11(4):59–71, 1990.

[15] B. Chandrasekaran. Models vs Rules, Deep vs Compiled, Content vs Form: Some Distinctions in Knowledge Systems Research. *IEEE Expert*, To Appear(4):0, 1991.

[16] B Chandrasekaran. Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design. *IEEE Expert*, 1(3):23–30, 1986.

[17] B. Chandrasekaran. Towards a functional architecture for intelligence based on generic information processing tasks. In *Proceedings of the International Joint Conference on Artificial Intelligence*. IJCAI, 1987.

[18] B. Chandrasekaran. What kind of information processing is intelligence? a perspective on ai paradigms and a proposal. In D. Partridge and Y. Wilks, editors, *Source Book on the Foundations of AI*. Cambridge University Press, 1987.

[19] B. Chandrasekaran. Generic tasks as building blocks for knowledge-based systems: The diagnosis and routine design examples. *Knowledge Engineering Review*, In Press 1989.

[20] B. Chandrasekaran, J. Smith, and J. Sticklen. "Deep" Models and their relation to diagnosis. In Furukawa, editor, *Medical Applications of AI*. Science Press, Amsterdam, 1987. Invited paper, Toyobo Foundation Symposium on Artificial Intelligence in Medicine, Tokyo, Japan, August 1986.

[21] B. Chandrasekaran, A. Goel, and D. Allemang. Connectionism and information processing abstractions. In *Proceedings of the AAAI Symposium on Parallel Models of Intelligence: How Can Slow Components Think So Fast?*, pages 66–85, Stanford University, March 1988.

[22] B. Chandrasekaran and Ashok Goel. From numbers to symbols to knowledge structures: Pattern recognition and artificial intelligence perspectives on the classification task. *IEEE Transactions on Systems, Man and Cybernetics*. To appear, 1988.

[23] B. Chandrasekaran and W. Punch III. Data validation during diagnosis, a step beyond traditional sensor validation. In *Sixth National Conference on Artificial Intelligence*. The Ohio State Univeristy, 1987.

[24] B. Chandrasekaran, John Josephson, and David Herman. The generic task toolset: High level languages for the construction of planning and problem solving systems. In *Proceedings Workshop on Space Telerobotics*, NASA and Jet Propulsion Laboratories, Pasadena, California, 1987.

[25] B. Chandrasekaran, J. Smith, and J. Sticklen. "Deep" Models and their relation to diagnosis. In Furukawa, editor, *Medical Applications of AI*. Science Press, Amsterdam, 1987. Invited paper, Toyobo Foundation Symposium on Artificial Intelligence in Medicine, Tokyo, Japan, August 1986.

[26] B. Chandrasekaran, Michael C. Tanner, and John R. Josephson. Explaining control strategies in problem solving. *IEEE Expert*, Spring:9–24, 1989.

[27] J. Sticklen B. Chandrasekaran and W. Bond. Distributed causal reasoning. *Knowledge Acquisition*, 1:139–162, 1989.

[28] Ashok Goel and Tom Bylander Computational feasibility of structured matching. Technical report, The Ohio State University, 1989.

[29] Ashok Goel and B. Chandrasekaran. Integrating model-based reasoning and case-based reasoning for design problem solving. Technical report, The Ohio State University, 1988.

[30] Ashok Goel and B. Chandrasekaran. Understanding device feedback. Laboratory for artificial intelligence research, The Ohio State University, Department of Computer and Information Science, March 1988.

[31] Ashok Goel, B. Chandrasekaran, and Donald Sylvan. JESSE: an information processing model of policy decision making. In *Proceedings of the Third Annual Expert Systems in Government Conference*, pages 178–18, Washington, D. C., October 19-23 1987. Sponsored by the IEEE Computer Society.

[32] Ashok Goel, John Kolen, and Dean Allemang. Learning in connectionist networks: Has the credit assignment problem been solved? In James Johnson, editor, *Proceedings of the Aerospace Applications of Artificial Intelligence Conference*, Dayton, Ohio, October 1988. Conference sponsored by ACM SIGART.

[33] Ashok Goel, J. Ramanujam, and P. Sadayappan. Towards a neural architecture for abductive reasoning. In *Proceedings of the Second IEEE International Conference on Neural Networks*, volume II, pages 681–688, San Diego, California, July 24-27 1988.

[34] Ashok Goel, P. Sadayappan, and John Josephson. Concurrent synthesis of composite explanatory hypotheses. In David Bailey, editor, *Proceedings of the Seventeenth International Conference on Parallel Processing: Algorithms and Applications*, volume III, pages 156–160, St. Charles, Illinois, August 15-19 1988.

[35] Ashok Goel, N. Soundararajan, and B. Chandrasekaran. Complexity in classificatory reasoning. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 421–425, Seattle, Washington, July 13-17, 1987.

[36] Ashok Kumar Goel. Integration of Case-Based Reasoning for Adaptive Design Problem Solving. Dissertation, Advisor: B. Chandrasekaran, The Ohio State University, 1989.

[37] D. Herman and D. Brown. DSPL: language for routine design. *Applied Artificial Intelligence Reporter*, 4(7):8–9,14–19, April-July 1987.

[38] D. J. Herman. Dspl++, a high-level language for building design expert systems with flexible use of multiple methods. Ph. D. dissertation, Computer & Information Science, The Ohio State University, in preparation., 1990.

[39] Todd R. Johnson, Jr. Jack W. Smith, and B. Chandrasekaran. Generic tasks and Soar. In *AAAI Spring Symposium*, 1989.

[40] J. Josephson, B. Chandrasekaran, J. Smith, and M. Tanner. A mechanism for forming composite explanatory hypotheses. *IEEE Trans. System, Man & Cybernetics*, pages 445–454, May/June 1987.

[41] John Josephson, Diana Smetters, and Others. The Generic Task Toolset, Fafner Release 1.0, Introduction. Laboratory for AI Research Technical Report, 1989.

[42] John R. Josephson, Diana Smetters, Richard Fox, Dan Oblinger, Arun Welch, and Gayle Northrup. Integrated generic task toolset: (Fafner Release 1.0): introduction and user's guide.

[43] J Sticklen, G. P. Robertson, and R. T. Tufankji-Attar. An artificial intelligence-based approach to modeling an ecological problem: A methodology to capture and extend understanding of the global n2o cycle. In *Proceedings of the Summer Computer Simulation Conference*, pages 454–459, 1989.

[44] Anne M. Keuneke. Machine Understanding of Devices Causal Explanation of Diagnostic Conclusion. Dissertation, Advisor: B. Chandrasekaran, The Ohio State University, 1989.

[45] Dale Moberg and John Josephson. Diagnosing and fixing faults in theories: Appendix: An implementation note. In *Computational Models of Discovery and Theory Formation*, 1990.

[46] D.R. Myers, J.F. Davis, and D. Herman. A task oriented approach to knowledge-based systems for process engineering design. *Computers and Chemical Engineering*, 12(9/10):959–971, August 1988.

[47] Hari Narayanan and B. Chandrasekaran. Reasoning Visually about Spatial Interactions. LAIR Technical Report, 1991.

[48] William F. Punch. A Diagnosis System Using a Task Integrated Problem Solver Architecture (TIPS), Including Causal Reasoning. Dissertation, Advisor: B. Chandrasekaran, The Ohio State University, 1989.

[49] Integrating classification-based compiled level reasoning with functional-based deep level reasoning. Laboratory for artificial intelligence research, The Ohio State University, Department of Computer and Information Science, 1989.

[50] J. Sticklen, B. Chandrasekaran, and W. Bond. Distributed causal reasoning. Knowledge Acquisition, 1989.

[51] Michael C. Tanner. Explaining Knowledge Systems: Justifying Diagnostic Conclusions. Dissertation, Advisor: B. Chandrasekaran, The Ohio State University, 1989.

Appendix

Dated July 26, 1990

To Appear in AI Magazine

Design Problem Solving: A Task Analysis[1]

B. Chandrasekaran

Laboratory for AI Research
Department of Computer & Information Science
The Ohio State University
Columbus, OH 43210
USA
Electronic mail: chandra@cis.ohio-state.edu

[1]This work has evolved over a number of years. Earlier versions have appeared as Chapter 2 of (Brown and Chandrasekaran, 1989) and in *Research in Engineering Design, 1989, 1:75-86.*

Abstract.

I propose a task structure for design by analyzing a general class of methods that I call "Propose-Critique-Modify" methods. The task structure is constructed by identifying a range of methods for each task. For each method, the knowledge needed and the subtasks that it sets up are identified. This recursive style of analysis provides a framework in which we can understand a number of particular proposals for design problem solving as specific combinations of tasks, methods and subtasks. Most of the subtasks are not really specific to design as such. The analysis shows that there is no one ideal method for design, and good design problem solving is a result of recursively selecting methods based on a number of criteria including knowledge availability. How the task analysis can help in knowledge acquisition and system design is discussed.

The Task Structure Methodology

Design problem solving is a complex activity involving a number of subtasks, and a number of alternative methods potentially available for each subtask. The structure of tasks has been a key concern of recent research in task- oriented methodologies for knowledge-based systems (Chandrasekaran, 1986; Clancey, 1985; Steels, 1990; McDermott, 1988). One way to conduct a task analysis is to develop a *task-structure* (Chandrasekaran, 1989) that lays out the relation between a task, applicable methods for it, the knowledge requirements for the methods and the subtasks set up by them. The major goal of this paper is to develop a task structure for design as a knowledge-based problem solving activity.

Design as Search in a Space of Subassemblies

Designing artifacts that are meant to achieve some functions within some constraints is an important class of design with characteristic properties (Goel and Pirolli, 1989). We concentrate on this class of design problems in this paper.

For sufficiently complex versions of the design problem a common theme emerges for design as a process: it involves mappings from the space of design specifications to the space of devices or components (often referred to as mapping from behavior to structure), typically conducted by means of a search or exploration in the space of possible subassemblies of components. This is in fact the origin of the frequent suggestion that design is a *synthetic* task.

The design problem is formally a search problem in a very large space for objects that satisfy multiple constraints. Only a vanishingly small number of objects in this space constitute even "satisficing," not to speak of optimal, solutions. What is needed to make design practical are strategies that radically shrink the search space.

Set against the view of design as a deliberative problem solving process is the view of design as an "intuitive," almost instantaneous, process where a design solution comes to the mind of the designer. Artistic creations and scientific theories are often said by their creators to have occurred to them *in this manner*. Even when a plausible solution occurs in this way, the proposal still needs to be evaluated, critiqued and modified by deliberatively examining alternatives. That is, except in simple cases, deliberative processes are still essential for real-world design.

Functions, Constraints, Components and Relations

A designer is charged with specifying an artifact that delivers some functions and satisfies some constraints. For each design task, the availability of a (possibly large and generally only implicitly specified) set of ***primitive components*** can be assumed. The domain also specifies a repertoire of ***primitive relations*** or connections that are possible between components. An electronics engineer, for example, may assume the availability of transistors, capacitors, and other electrical components when he is designing a waveform generator. Primitive relations in that domain are serial and parallel connections between components.

Of course, design is in general recursive: if a certain component that was assumed to be available is in fact not available, the design of that can be

undertaken in the next round. However, the vocabulary of primitive components and relations may be rather different than those for the original device.

Functions can be expressed as a state or a series of states that we want the device to achieve or avoid under specified conditions. Functions may be explicitly stated as part of problem specifications, or they may be implicit in the designer's understanding of the domain. An example of an implicit function in many engineering devices is safety: e. g. a subsystem's role may only be explained as something that prevents leakage of a potentially hazardous substance, and this funtion may never be stated explicitly as part of the design specification (Keuneke, 1989).

Usually, design specifications will mention, in addition to desired functionalities, a number of constraints[2]. The distinction between functions and constraints is hard to pin down formally; functions *are* constraints on the behavior or properties of the device. It is, however, useful to distinguish functions from other constraints, since the former are the primary reason why the device is desired. Design constraints can be on the properties of the artifact (e. g., "Should not weigh more than ..."), on the process of making the artifact from its description (manufacturability constraints), on the design process itself (e. g., "I want a design within a week") and so on. A computationally effective process of design is to generate a candidate design based on functions and then to modify it to meet the constraints.

Definition of the Design Task

Consider the following definition of the design task.

> The design problem is specified by:
>
> - a set of functions (explicitly stated by the design consumer as well as implicit ones defined by the domain) to be delivered by an artifact and a set of constraints to be satisfied, and

[2]The constraints that are described as part of the design specification ought to be distinguished from the term "constraint" that appears in description of design methods, such as "constraint-directed problem solving".

- a "technology", i. e., a repertoire of components assumed to be available and a vocabulary of relations between components.

The constraints may pertain to the design parameters themselves, to the process of making the artifact, or to the design process. The solution to the design problem consists of a complete specification of a set of components and their relations which together describe an artifact that delivers the functions and satisfies the constraints. The solution is expected to satisfy a set of implicit criteria as well, e. g., it is not much more complex or costly than plausible alternatives (ruling out Rube Goldberg devices).

The preceding definition also captures the *domain-independent* character of design as a generic activity. Planning, programming and engineering design all share the above definition, as well as many of the subprocesses, to a significant degree. Nevertheless, there are versions of the design problem for which the above definition needs to be modified or extended. Examples are:

- At the start of the design process only a minimal statement of functions and constraints may be available, and additional ones may be developed in parallel with the design process itself.
- Some design problems involve extensive trade-off studies, where a part of the design process is search for ways in which the functions or the constraints may be relaxed or otherwise modified.
- "Tinkering" is a time-honored method of invention where the design space is being explored without any specific set of functions in mind. Functions may be identified for structural configurations that arise during exploration.
- The world of primitive objects may be very open-ended, and only implicitly specified.

The design framework that I will be presenting can be extended to cover these variations.

The Task-Structure

Let us say we have a problem solving task[3] T, and let M be some *method* suggested for the task. A method can be described in terms of the *operators* it employs, the *objects* that it operates on, and any additional knowledge about how to organize operator application to satisfy the goal. At the knowledge level, the method is characterized by the knowledge the agent needs to set up and apply the method. Different methods for the same task may call for knowledge of different types.

To take a simple example, for the task of multiplying two multi-digit numbers, the "logarithmic method" consists of the following series of operations: *extract the logarithm* of each of the input numbers, *add* the two logarithms, and *extract the anti-logarithm* of the sum. (The operators are italicized. Their arguments as well as the results are the objects of this method.)

Note that one does not typically include, at this level of description of the logarithmic method, specifications about how to extract the logarithm or the anti-logarithm, or how to do the addition. If the computational model does not provide these capabilities as primitives, performing these operations can be set up as *subtasks* of the method. Thus, given a method, applying any of the operators of a method can be set up as a subtask. Some of the objects a method needs may be generic to a class of problems in a domain. As an example, consider hierarchical classification using a malfunction hierarchy, a common method of diagnosis. Operations of "Establish-Hypothesis" and "Refine-Hypothesis" are applied to the hypotheses in the hierarchy. These objects are useful to solve many instances of the diagnostic problem in the domain. If the malfunction hypotheses are not directly available, generation of such hypotheses can be set up as subtasks. A common method for the generation of such objects is compilation from so-called "deep" knowledge. Structure-function models of the device that is being diagnosed have been proposed and used as deep models to generate malfunction hypotheses (Chandrasekaran, et al, 1988).

There is no finite set of mutually distinct methods for a task, since there

[3]In this paper, I use the terms "task" and "goal" interchangeably.

can be numerous variants on a method. Nevertheless, the term "method" is a useful shorthand to refer to a set of related proposals about organizing computation.

Types of Methods. One type of method is of particular importance in knowledge-based systems: methods which can be viewed as a problem space search (Newell, 1980). Designer-Soar (Steier, 1989) and AIR-CYL (Brown and Chandrasekaran, 1989) are examples of design systems which explore search spaces. For example, AIR-CYL can be understood as searching in a space of parameters for the components of an air-cylinder by using design plans which propose and modify parameter values.

Another class of methods consists of algorithms which directly produce a solution without any search in a space of alternatives, e. g., producing a set of design parameters by numerically solving a set of simultaneous equations. Such algorithms are only available for so-called well-structured problems[4]. Most real-world problems are ill-structured, and the role of domain knowledge is to help set up spaces of alternatives and to help control the search in those spaces.

A task analysis of this type can be continued recursively until methods whose operators are all directly achievable (within the analysis framework) are reached. In the following task analysis for design, I will explicitly indicate as subtasks only those to which I want to draw specific attention in my discussion. Other operators may exist which require additional problem solving as well.

[4]I subscribe to the view that such algorithms are simply degenerate cases of search where the agent has sufficient knowledge to make the correct choice at each choice point. But, pragmatically speaking, it is best to think of algorithmic methods as a separate type, since implementing them does not require supporting search in general.

A Task-Structure for Design: The Propose-Critique-Modify Family of Methods

The most common top level family of methods for design can be characterized as *Propose-Critique-Modify* (PCM) methods. These methods have the subtasks of Proposal of partial or complete design solutions, Verification of proposed solutions, Critiquing the proposals by identifying causes of failure, if any, and Modification of proposals to satisfy design goals. These subtasks can be combined in fairly complex ways, but the following is one straightforward way in which a PCM method may organize and combine the subtasks.

Example PCM Method:

- Step 1. Given design goal, Propose solution. If no proposal, exit with failure.
- Step 2. Verify proposal. If verified, exit with success.
- Step 3. If unsuccessful, Critique proposal to identify sources of failure. If no useful criticism available, exit with failure.
- Step 4. Modify proposal and return to 2.

While all of the PCM methods will need to have some way to achieve the iteration in Step 4 above, there can be numerous variants on the way the methods in this class work. For example, a solution may be proposed only for a part of the design problem, a part deemed to be crucial. This solution may then be critiqued and modified. This partial solution may generate additional constraints, leading to further design commitments. Thus subtasks can be scheduled in a fairly complex way, with subgoals from different methods alternating. It is hard to identify a separate method for each such variation. The implications of this for a design architecture are discussed in the concluding sections of the paper.

In this paper most of the attention is devoted to the Proposal subtask, since most of the design knowledge, per se, is used in this subtask. Every

task has a default method: one which uses compiled knowledge to get a solution without any problem solving. This method is practical only in simple cases. Because this method is potentially applicable to simple versions of all tasks, and has no interesting internal structure, I will not explicitly mention it in my discussion.

A task analysis should provide a framework within which various approaches to design can be understood. I will use selected examples of existing AI systems to illustrate the ideas, but there will be no attempt to provide a survey of all AI work on design.

Methods For Proposing Design Choices

Design proposal methods use domain knowledge to map part or all of the specifications to partial or complete design proposals. Three groups of methods can be identified:

- Problem decomposition/solution composition. In this class of methods, domain knowledge is used to map subsets of design specifications into a set of smaller design problems. Use of design plans is a special case of decomposition methods.
- Retrieval of cases from memory which correspond to solutions for design problems which are similar or "close" to the current problem.
- Family of methods that solve the design problem as a constraint satisfaction problem and use a variety of quantitative and qualitative optimization or constraint satisfaction techniques.

Decomposition and case-based methods help reduce the size of the search spaces, since the knowledge they use can be viewed as the compilation or chunking of earlier (individual or community) search in the design space. Conversion of a design problem into one amenable to global optimization algorithms requires substantial a priori knowledge of the structure of the design problem.

Decomposition/Solution Composition. I will treat this method in terms of all the features that an information processing analysis calls for:

types of knowledge and information needed and the inference processes that operate on this form of knowledge.

Knowledge needed is of the form D -> D1, D2,... Dn, where D is a given design problem, and Di's are "smaller" subproblems (i. e., associated with smaller search spaces than D). A number of alternate decompositions for a problem may be available, in which case a selection needs to be made, with the attendant possibility of backtracking and making another choice. Repeated applications of the decomposition knowledge produce *design hierarchies*. In well-trodden domains, effective decompositions are known and little search for decompositions needs to be conducted as part of routine design activity. For example, in automobile design, the overall decomposition has remained largely invariant over several decades.

Decomposition knowledge in design generally arises when the functional specifications can be decomposed into a set of subfunctions (Freeman and Newell, 1971). Design decomposition knowledge may come in the form of part-subpart decomposition, if a direct mapping is available between functions and components.

The following are two important subgoals of the Decomposition Solution Composition method.

- Generating specifications for subproblems. The functional and other specifications on D need to get translated into specifications for each of the subproblems D1,... Dn.
- Gluing the subproblem solutions into a solution to the original design problem.

In most routine design, these subtasks are not explicit: they are either solved by compiled knowledge or the problem specification already implies a solution to these problems. In the general case, however, additional problem solving is needed.

How a Decomposition/Solution Composition method might actually organize and use the subgoals is given by the following example.

Example Problem Decomposition/Solution Recomposition Method:

- Step 1. (Search in the space of decompositions.) Choose from among alternative decompositions for the given design problem D.
- Step 2. Generate specifications for subproblems in the chosen decomposition.
- Step 3. Set up each subproblem as a design problem. Solve them in some order determined by control strategies and other domain knowledge (e.g., progressive deepening)
- Step 4. If subproblems solved, Recompose solutions of subproblems into solution for D, and exit.
- Step 5. If failure in Steps 3 or 4, go to Step 1 to make another choice, or relax specifications and go to Step 2.

All the caveats mentioned in connection with the PCM method earlier apply. Specifically, control of how subproblems are solved may be quite variable and more complex than indicated above. Some of the sources of this complexity are discussed below.

Given a design problem, it may not always be possible to generate all the constraints for its subproblems from the original problem's specifications alone. In many domains, constraint generation for some subproblems alternates with partial design of others, which in turn provides additional information for constraints for yet other subproblems. There may be a complex process of commitments and backtracking. In extreme cases, most of design problem solving may consist of search for parameters that make all the subproblems solvable. For example, the *Propose and Revise* method (Marcus, 1985) involves making commitments to some subparts of the design problem (Propose part) and then Revise these when some constraints for other parts of the problem are violated.

In configuration tasks (Mittal and Frayman, 1989), subproblem solutions are given as part of the problem (i.e, the desired functions are mapped into a set of key components), and the remaining task is dominated by the

subtasks of specification generation and solution recomposition. In order for components A and B to be connected, certain pre- and post-conditions may need to be satisfied. If these conditions are not available *a priori*, they need to be derived from configuration behaviors. Discovery of connection conditions and checking of whether specific configuration proposals result in desired functional behaviors can often use simulation as a problem solving method (e.g., (Kelly, 1982)).

There can be complex dependencies between constraints among subproblems. In situations where not only are commitments for D1 going to constrain the specifications for D2,.. Dn, but the commitments for the latter may further specify constraints for D1 as well, a strategy that Steier (1989) has identified as *progressive deepening*, is a natural strategy to emerge. This strategy involves making some commitment for each subproblem at each pass, using these commitments to generate additional specifications, undoing earlier commitments as needed, and repeating this process.

Control Issues. There are two sets of control issues, one dealing with which sets of decompositions to choose (in Step 1 in the example Decomposition/Recomposition method above), and the other concerned with the order in which the subproblems within a given decomposition ought to be attacked (Step 3). For the first problem, in the general case, the decomposition will produce an AND or an OR node. The decompositions in an AND node will all need to be solved, while for an OR node only one of the decompositions will need to be solved. Finding the appropriate decomposition requires search in an AND/OR graph. But as a rule such searches are expensive. In domains where multiple decompositions are possible but there are no easily formalizable heuristics to choose among them, the machine may be effective in proposing alternatives while the human evaluates them and makes a selection.

In routine design extensive searches in the spaces of possible decompositions are avoided by limiting the number of possible decompositions at each choice point to one or very few. This leads to the availability of a design hierarchy for design in that domain.

Transformation methods (Balzer, 1981) for algorithm synthesis are

examples of decomposition methods. In this approach, a set of high-level specifications of an algorithm are converted into a series of programming language level commitments. This is done by mapping subsets of specifications into a "component" for which some implementation level commitments have been made. Each such commitment will typically constrain other implementation commitments. Because of this, search in the space of possible transformations may often be needed. In most implemented transformation systems, humans choose from a set of alternative transformations presented by the design system.

Regarding the order in which subproblems in a given decomposition are to be attacked, the main constraint is knowledge about dependencies between subproblems that I just discussed. When the subproblems are organized in the form of a design hierarchy, the default control is control *top-down*, but actual control can be complicated. For example, a component at the leaf level of the design hierarchy may be the most limiting component and many other components and subsystems can only be designed after that is chosen. Part of design process in this case will appear to have a bottom up flavor. In general, appropriate control strategies come about based on the dependencies between subproblems.

Design Plans. A special case of decomposition knowledge is *design plans*, representing a precompiled partial solution to a design goal (Rich (1981), Johnson (1985), Friedland (1979), Mittal (1986), Brown and Chandrasekaran (1989)). A design plan specifies a sequence of design actions to take for producing a design or part of a design. Design commitments made by a design plan may be abstract, i. e., choices are made not at the level of primitive objects, but rather intermediate level design abstractions which need to be further refined at the level of primitive objects. For example, in designing an automobile, a design plan may commit to choice of diesel engine as the power plant. While this is a design proposal in the sense that a commitment is being made, the diesel engine design itself is not specified in detail at this stage, but posed as a subtask to be solved by any of the available methods.

Thus a design plan D may set up other design problems D1, . . . , Dn as subproblems, and, in this sense, it is decomposition knowledge in a strong

form: how the main problem goals are transformed into goals to be allocated to subproblems and how the solutions to the subproblems are put back together for obtaining a solution to the original design problem are all directly encoded in the plan.

Design plans can be indexed in a number of ways. Two possibilities are to index by design goals (*for achieving* < *goal* > *use* < *plan* >), or by components (*for designing* < *part* >, *use* < *plan* >). Each goal or component may have a small number of alternative plans attached to them, with perhaps some additional knowledge that helps in choosing among them.

Control and inference issues in the use of plans are similar to those in the general case of decomposition: alternate plans are possible, and in routine design, design plan hierarchies may emerge. The default control strategy can be characterized as *instantiate and expand*. That is, the plan's steps specify some of the design parameters, and also specify calls to other design plans. Choosing an abstract plan and making commitments that are specific to the problem at hand is the instantiation process, and calling other plans for specifying details to portions is the expansion part.

A number of additional pieces of information may be needed or generated as this expansion process is undertaken. Information about dependencies between parts of the plan may need to be generated at runtime (e. g., discovering that certain parameters of a piston would need to be chosen before that of the rod), and some optimizations may be discovered at run time (e.g., the same base that was used to attach component A can also be used to attach component B). NOAH (Sacerdoti, 1975) is an early example of run-time generation of dependencies and optimization.

Design Proposal By Case Retrieval. A major source of design proposal knowledge is design cases, instances of successful past design problem solving. Cases can arise from an individual's problem solving experience or that of an organization such as a design firm or a design community. Cases can be episodic (i. e., represent one problem solving episode) or can represent the result of abstraction and generalization over several episodes. Design plans can be considered to be fairly abstracted versions of numerous cases.

Sussman (1973) proposed that a design strategy is to choose an already-completed design that satisfies constraints closest to the ones that apply to the current problem, and modify this design for the current constraints. Schank (1982) has emphasized the importance of case-based problem solving in general. Recent work on case-based reasoning in planning and design (Hammond, 1989; Goel and Chandrasekaran, 1989) explores this family of methods. In case-based reasoning, "almost correct designs" are obtained by searching a memory bank of previous cases for a design that is similar to the one that is currently being sought.

The heart of case-based design proposal is *Matching:* How to choose the design that is "closest" to the current problem? Clearly some features of the cases are more important in matching than others. Some notion of prioritizing over goals or differences, in the sense of *means-ends analysis*, may be needed.

Indexing of cases with a rich vocabulary of features of the case and the goals it satisfies is a key idea in case-based reasoning. Matching and retrieval can be driven by associative processes on these indices. Much of the work in case-based planning has used domain-specific goals to index cases. For the problem of designing engineering artifacts, the design cases need to be indexed in terms of the output behaviors of interest. For example, Goel and Chandrasekaran (1989b) propose that design cases be indexed using their functions. More generally, cases can be indexed by a causal representation that relates the structure of the device to its function, and show how this can help in retrieval. Goel (1989) has a proposal for how matching and retrieval can benefit from a principled representation for design goals and states for the device and the substances the device operates with.

Case-based design proposal has a lot in common with the use of analogical reasoning in design. Maher, Zhao and Gero (1988) propose that analogical reasoning in design is at the heart of design creativity.

Design Proposal by Constraint Satisfaction. Under fairly strong assumptions particular classes of design problems can be formulated as optimization, constraint satisfaction or algebraic equation solving problems. What is common to all these formulations is that the solution lies in a space determined by simultaneous constraints, and specific classes of

computational algorithms may be available to locate that space directly. In particular, when the structure of the design is already specified, but parameters are determined by the specifics of a design problem, numerical or symbolic optimization techniques may be useful for design proposal. Linear, integer and dynamic programming techniques have been used to solve design problems formulated in this manner.

Some versions of the constraint satisfaction problem can be solved by constraint propagation. Constraints can be propagated in such a way that the component parameters are chosen to incrementally converge on a set that satisfies all the constraints (Stefik, 1981).

Formally *all* design can be thought of as constraint satisfaction, and one might be tempted to propose global constraint satisfaction as a universal solution for design. But unless knowledge is employed to reduce the size of the space (such as by decomposing problems into smaller problems), design by constraint propagation can be computationally intractable. Knowledge such as decomposition can create subproblems with sufficiently small problem spaces in which constraint satisfaction methods can work without excessive search.

Verification

This subtask involves checking that the design proposal satisfies the functional and other specifications. There are two families of methods for this:

- Attributes of interest can be directly calculated or estimated by means of domain-specific algorithms or formulae (e.g., use of algebraic formula to calculate total weight or cost, or use of finite-element methods to calculate stress distribution). Direct calculation methods are not of much interest from an AI point of view.
- Behaviors of interest can be derived by *simulation*. These behaviors can be checked against requirements.

Simulation takes as input a description of the structure of the system and generates as output the behaviors of interest. The methods used in

simulation should mirror the rules by which the behavior of assemblages or components is composed from the properties of the components. There are quantitative simulation methods which use equations that directly describe the results of this composition. These equations again are domain-specific. For example, differential equations may be used to describe the behavior of a reaction in a reaction vessel. The structural description in a proposed design of a reaction vessel can be translated into parameters of the differential equation and the equation simulated to derive behaviors of interest.

There are generic AI techniques for generating behavior from structure that could be useful for simulation. Qualitative simulation (see Forbus, 1988, for a survey of the current state of the art), consolidation (Bylander, 1988) and functional simulation (Sticklen, 1989) are examples of AI techniques that are available for deriving behaviors given structure. A proposed design can be simulated under various input conditions and the behavior evaluated. All these techniques take as input a structural description and, using qualitaive descriptions of component behaviors and rules of composition, mimic the operation of the device to produce qualitative descriptions of behavior. Qualitative and quantitative simulation may alternate: a qualitative simulation may identify behaviors likely to be in unacceptable ranges and a more focused quantitative procedure may be used to get more precise values.

Visual simulations. Visual simulation of artifacts is widely used by human designers in verification. Designs are imagined, represented, and communicated pictorially in domains such as architecture and mechanical engineering. (See Goel and Pirolli for design protocol studies which show the prevalence of images during design.) It is clear that there is a need for pictorial representations and symbolic representations to coexist in design systems. A major use of imaginal representations is in simulation of design proposals, but they play a role as well in making design proposals by analogy with other domains. Little AI research has been done so far on visual representations that have the qualities needed for pictorial reasoning and imagination, and that also have the symbolic properties needed for arbitrary referencing and composition by parts. A beginning in this direction is proposed in (Chandrasekaran and Narayanan, 1990) and use of

such representations for simulation is discussed in (Narayanan and Chandrasekaran, 1990).

Critiquing

Critiquing is the subtask in which causes of failure of a design are analyzed: parts of the structure are identified as potentially responsible for the unacceptable behavior or constraint violation. Critiquing is really a generalized version of the diagnostic problem, i.e., a problem of mapping from undesirable behavior to parts of the structure responsible for the behavior. Modification of design can be directed to these candidates. Of course localization of responsibility for failure will not always work: the entire approach to the design may need to be changed.

What is needed for criticism is information about how the structure of the device contributes to (or is intended to contribute to) the desired overall behavior. An AI method that is commonly used for this subtask is dependency analysis (Stallman and Sussman, 1977). This method is applicable if explicit information is available in the form of dependencies, i.e., knowledge that explicitly relates types of constraint or specification violations to prior design commitments. For example, if total weight of a proposed design is higher than the weight limit, domain-specific knowledge is usually available which identifies parts whose weights are both sufficiently large and can be adjusted. Dependencies may be discovered by analyzing pre- and post-conditions of design operators. For example, if a certain output behavior (say, voltage in an electronic device) of a proposed design is excessive, the inputs the output stage can be traced back to identify which of the components upstream may have contributed to the specific output. This analysis may use simulation as a subtask.

Most of the proposals for critiquing that have been in the case-based reasoning literature use domain-specific critics and are variations on pre-compiled patterns of relating output behavior to possible changes. The approach of Goel (1989) for critiquing a design proposal is based on a functional analysis of the proposed design. If a design proposal is endowed with causal indices that explicitly indicate the relation between structure and intended functions, then it is relatively easy to identify substructures for modification (Goel and Chandrasekaran, 1989).

Modification

Modification as a subtask takes information about failure of a candidate design as its input and then changes the design so as to get closer to the specifications. Basically, what is required is changing a functional subpart of the proposed design, or adding components to the proposed design, so as to satisfy the design specifications. Depending upon the sophistication about failure analysis and other forms of knowledge available, a number of problem-solving processes are applicable. Some of them are briefly outlined in the following paragraphs.

Modification may be driven by a form of means-ends reasoning, where the differences are "reduced" in order of most to least significant. Especially useful here is knowledge that relates the desired changes in behavior to possible structural changes (Goel, 1989).

A related search approach is one where modification is done by some form of hill-climbing. In this method, parameters are changed, direction of improvement noted, and additional changes are made in the direction of maximal increment in some measure of overall performance. This is especially applicable where the design problem is viewed as a parameter choice problem for a predetermined structure (e.g., the Dominic system (Dixon, 1984)).

Modification is straightforward in dependency-directed methods. Once the dependency point is reached by back-tracking, simply an alternative choice is made from the list of finite choices available.

Some systems that perform routine design problems have explicit knowledge about what to do under different kinds of failures. This information can be attached to the design plans (DSPL, Brown and Chandrasekaran, 1989).

Criticism may reveal the need to add new functions. If these functions can be added modularly, i.e., by the creation and integration of separate substructures that deliver the functions, the design of the additional structures can be viewed simply as new design problems to be solved by all the methods available for design. The subtasks of generation of specifications for these additional design problems and integrating their

solutions were discussed in the section on problem decomposition and solution recomposition.

Discussion of the Task Structure

The task-structure for design described in the preceding sections[5] is summarized in tabular form in Table 1. A task-structure is a description of the task, proposed methods for it, their internal and external subtasks, knowledge required for the methods, and any control strategies for the method. Thus the task analysis provides a clear road map for knowledge acquisition. How the analysis can be used to integrate the methods and goals is discussed in the following section.

Choice of methods. How are methods to be chosen for the various tasks? The following is a set of criteria:

- Properties of the solution. Some methods may produce answers which are precise, while answers of the others may only be qualitative. Some of them may produce optimal solutions, while others may produce satisficing ones.
- Properties of the solution process. Is the computation pragmatically feasible? How much time does it take? Memory?
- Availability of knowledge required for the method to be applied. For example, a method for design verification might require that we have available a description of the behavior of the device as a system of differential equations; if this information is not available directly and if it cannot be generated by additional problem solving, the method cannot be used.

A delineation of the methods and their properties helps us to move away from abstract arguments about ideal methods for design. Each method in a

[5]The task structure described here is inherently incomplete: additional methods may be identified for any subtask as a result of further research.

task-structure can be evaluated for appropriateness in a given situation by asking questions reflecting the above criteria. While some of this evaluation can take place at problem solving time, much of it can be done at the time of design of the knowledge system; this evaluation can be used to guide a knowledge system designer in the choice of methods to implement.

Different types of methods may be used for different subtasks. For example, a design system may use a knowledge-based problem solving method for the subtask of creating a design, but use a quantitative method such as a finite element method for the subtask of evaluating the design.

Implications for an Architecture for Design Problem Solving

Because of the multiplicity of possible methods and subtasks for a task, a task-specific architecture that is exclusively for design is not likely to be complete: even though design is a generic activity, there is no one generic method for it. Further, note that subtasks such as simulation are not particularly specific to design as a task. Thus if the knowledge for these modules is embedded within a design architecture, either they will be unavailable for other tasks which require simulation as a subtask, or the knowledge for these tasks will need to be replicated. Thus instead of building monolithic task-specific architectures for such complex tasks, a more useful architectural approach is one that can invoke different methods for different subtasks in a flexible way.

Following the ideas in the work on task-specific architectures, we can support methods by means of special-purpose shells that can help encode knowledge and control problem solving. This is an immediate extension of the generic task methodology (Chandrasekaran, 1986). These methods can then be combined in a domain-specific manner, i. e., methods for subtasks can be selected in advance and included as part of the application system. Or, methods can be chosen at run-time for the tasks recursively, based on the criteria listed above in the paragraph on choice of methods. For the latter, what is needed is a task-independent architecture with the capability of evaluating different methods, choosing one, executing it, setting up subgoals as they arise from the chosen method and repeating the process. Soar (Rosenbloom, Laird, and Newell, 1987), BB1 (Hayes-Roth, 1987) or

TIPS (Punch, 1989) are good candidates for such an architecture. This approach combines the advantages of task-specific architectures and the flexibility of run-time choice of methods. The DSPL++ work of (Herman, 1990) is an attempt to do precisely this.

Using method-specific knowledge and strategy representations within a general architecture that helps select methods and set up subgoals is a good first step in adding flexibility to the advantages of the task-specific architecture view. However, it can have limitations as well. For many real world problems, switching between methods may result in control that is too large-grained. In order to see this, consider my earlier description of a PCM method. The method description calls for a specific sequence of how the operators of Propose, etc. were to be applied. As pointed out in my discussion on the PCM method, numerous variants of the method, with complex sequencing of the various operators, may be appropriate in different domains. It would be a hopeless task to try to support all these variants of the methods by method-specific architectures or shells. It is much better in the long run to let the task-method-subtask analysis to guide us in the identification of the needed task-specific knowledge and let a flexible general architecture determine the actual sequence of operator application by using additional domain-specific knowledge. The subtasks can then be combined flexibly in response to problem solving needs, achieving a much finer-grained control behavior. (See Johnson, Smith & Chandrasekaran, 1989 for realizing generic task ideas in Soar.)

The task structure also makes clear how "AI-like" methods and other algorithmic or numerical methods can be flexibly combined, much as human designers alternate between problem solving in their heads and formal calculations. For example, a designer may need to make sure that the maximum current in a proposed circuit is less than the limits for its components, and at that point, he may set up current and voltage equations and solve them. If he finds that the current in one branch of the circuit is more than the permitted limit, he may go back to critiquing the design to look for possible places to change the design. The task-structure view that I have outlined shows how computer-based design systems can also similarly engage in a flexible integration of problem-solving and other forms of algorithmic activity. The key is that the the top-level control is

REFERENCES:

Balzer, R. 1981. Transformation Implementation: an Example. *IEEE Trans. Software Engineering*, Vol. SE-7, pp. 3-14.

Brown, D. C. and Chandrasekaran, B. 1989. *Design Problem Solving: Knowledge Structures and Control Strategies*. Morgan Kaufmann, San Mateo, California.

Bylander, T. C. 1988. A Critique Of Qualitative Simulation From a Consolidation Point Of View, *IEEE Systems, Man & Cybernetics*, 18 (2), pp. 252-268.

Chandrasekaran, B. 1986. Generic Tasks in Knowledge-Based Reasoning: High- Level Building Blocks for Expert System Design, *IEEE Expert*, 1 (3), pp. 23-30.

Chandrasekaran, B. 1989. Task Structures, Knowledge Acquisition and Learning, *Machine Learning*, 4, 339-345.

Chandrasekaran B. and Narayanan, N. H. 1990. Integrating Imagery and Visual Representations, *Proc. Cognitive Science Society Annual Conference*, MIT, Cambridge, MA.

Clancey, W. J. 1985. Heuristic Classification:, *Artificial Intelligence* 27 (3), 289-350.

Dixon, J. R., Simmons, M. K. and Cohen, P. R. 1984. An Architecture for Application of Artificial Intelligence to Design, *Proceedings of the 21st Design Automation Conference, IEEE*, pp. 634-640.

Forbus, Kenneth D. 1988. Qualitative Physics: Past, Present and Future, *Exploring Artificial Intelligence*, Morgan Kauffman, San Mateo, CA, pp. 239-296.

Friedland, P. 1979. Knowledge-Based Experimental Design in Molecular Genetics, *Proceedings of the 6th International Joint Conference in Artificial Intelligence, IJCAI*, Tokyo pp. 285-287.

Goel, A. 1989. Integration of Case-Based Reasoning and Model-Based

goal-oriented, and it can set up subgoals and choose methods that are appropriate to the subgoal. If the appropriate method for a subtask is a numerical algorithm, that method can be invoked and executed, at the end of which control reverts to the top-level for pursuing other goals.

Concluding Remarks

Over the last several years, there have been a number of working systems which perform some version of the design task in some domain. These design proposals do not always bring out what is common among the different tasks of design. There have also been attempts to develop formal "first principles" algorithms for design that are meant to cover all types of design. Such general algorithms are, however, computationally intractable, and are not particularly helpful in identifying the sources of power and tractability in human design problem solving in most domains.

The view elaborated here is that there is a generic vocabulary of tasks and methods that are part of design, and that design problems in different domains simply differ in the mixture of subtasks and methods. Expertise, i.e., methods, and knowledge and control strategies for them, emerge over a period in different domains so as to help solve the task in a given domain tractably. The key to understanding all this is thus not in a uniform algorithm for design, but in the structure of the task, showing how the tasks, methods, subtasks and domain knowledge were related. The analysis also clarifies the relationship between task-specific architectures and more general-purpose architectures for knowledge systems.

ACKNOWLEDGEMENTS: Many ideas from my collaborations with the following have found their way into this paper: with David C. Brown and Ashok Goel on design problem solving, and with Tom Bylander, John Josephson, Todd Johnson, Jack W. Smith and Jon Sticklen on generic tasks. I am thankful to John Gero, Ashok Goel, Mary Lou Maher, Dale Moberg and David Steier for very useful comments on earlier drafts. The usual caveat holds good that they don't necessarily agree with all of what I am saying in the paper. Support from AFOSR (grants 87-0090 and 89-0250) and DARPA (contracts F30602-85-C-0010 and F49620-89-C-0110) is gratefully acknowledged.

Reasoning for Adaptive Design Problem Solving. Ph. D. dissertation. Computer & Information Science. The Ohio State University.

Goel, A. and Chandrasekaran, B. 1989. Functional Representation of Designs and Redesign Problem Solving. *Proc. Eleventh Intern. Joint Conf. on AI*. pp. 1388-1394.

Goel, A. and Chandrasekaran, B. 1989b. Use of Device Models in Adaptation of Design Cases. *Proceedings of the Second DARPA Case-Based Reasoning Workshop*. Pensacola, May 1989, pp. 100-109.

Goel, V. and Pirolli, P. 1989. Motivating the Notion of Generic Design within Information-Processing Theory: The Design Problem Space. *AI Magazine*. 10:1, pp. 18-38.

Hammond, K. 1989. *Case-Based Planning: Viewing Planning as a Memory Task*. Academic Press, Boston, MA.

Hayes-Roth, B. 1985. A Blackboard Architecture for Control. *Artificial Intelligence*. 26: 251-321.

Herman, D. J. 1990. DSPL++: A High-Level Language for Building Design Expert Systems with Flexible Use of Multiple Methods (tentative title). Ph. D. dissertation. Computer & Information Science. The Ohio State University.

Johnson, L. and Soloway, E. 1985. PROUST: Knowledge-Based Program Under- Standing, *IEEE Trans. Software Engineering*, 11 (3), pp. 267-275.

Johnson, T., Chandrasekaran, B., & Smith, J. W., Jr. 1989. Generic Tasks and Soar, Working Notes of the AAAI Spring Symposium on Knowledge System Development Tools and Languages. Stanford, CA. pp. 25-28.

Kelly, V. E. and Steinberg, L. I. 1982. The CRITTER System: Analyzing Digital Circuits by Propagating Behaviors and Specification. *Proceedings AAAI Conference, AAAI*, p. 284.

Keuneke, A. 1989. Machine Understanding Of Devices: Causal Explanations of Diagnostic Conclusions. Ph. D. dissertation, Computer &

Information Science, The Ohio State University.

Maher, M. L., Zhao, F., and Gero, J. S. 1988. Creativity in Humans and Computers, *Preprints Knowledge-Based Design in Architecture*, Gero, J. S. and Oksala, T. (eds.) Helsinki University of Technology, Helsinki, pp. 29-44.

Marcus, S., McDermott, J. and Wang, T. 1985. Knowledge Acquisition for Constructive Systems, *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI*, pp. 637-639.

McDermott, J. 1988. A Taxonomy of Problem-Solving Methods. In *Automating Knowledge Acquisition for Expert Systems*, Marcus, S., (ed.) Kluwer, Boston, pp. 225-256.

Mittal, S., Dym, C. and Morjaria, M. 1986. PRIDE: An Expert System for the Design of Paper Handling Systems, *IEEE Computer*, 19 (7), pp. 102-114.

Mittal, S. and Frayman, F. 1989. Towards a Generic Model of Configuration Tasks, *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, Michigan.

Narayanan, N. H. and Chandrasekaran, B. 1990. Qualitative simulation of spatial mechanisms: A preliminary report, TR, Laboratory for AI Research, Ohio State University. To be presented at the *AAAI, 1990, Workshop on AI and Simulation*.

Newell, A. 1980. Reasoning, Problem Solving and Decision Process: The Problem Space as a Fundamental Category, *Attention and Performance, VIII*, Lawrence Erlbaum, pp. 693-718.

Punch, W. 1989. A Diagnosis System Using a Task-Integrated Problem Solver Architecture (TIPS), Including Causal Reasoning," Ph. D. dissertation, Computer & Information Science, The Ohio State University.

Rich, C. 1981. A Formal Representation for Plans in the Programmer's Apprentice, *Proceedings of the 7th International Joint Conference on Artificial Intelligence, IJCAI*, Vancouver, B.C., Canada, pp. 1044-1052.

Rosenbloom, P. S., Laird, J. E. and Newell, A. 1987. SOAR: An Architecture for General Intelligence. *Artificial Intelligence*. 33 1-64

Sacerdoti, E. D. 1975. A Structure for Plans and Behavior. Technical Report 109, AI Center, SRI, Menlo Park, California.

Schank, R. 1982. Dynamic Memory: A Theory of Learning in Computers & People. Cambridge University Press, New York.

Stallman, R. and Sussman, G. 1977. Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis. *Artificial Intelligence*, 9, pp. 135-196.

Steels, L. 1990. Components of Expertise. *AI Magazine*, 11 (2), pp. 28-49.

Stefik, M. 1981. Planning with Constraints. *Artificial Intelligence*, 16, pp. 111-140.

Steier, D. 1989. Automating Algorithm Design Within an Architecture for General Intelligence. Ph. D. dissertation. School of Computer Science. Carnegie Mellon University. CMU-CS-89-128.

Sticklen, J. 1987. MDX2: An Integrated Medical Diagnosis System. Ph. D dissertation. Computer & Information Science. The Ohio State University

Sussman, G. J. 1973. A Computational Model for Skill Acquisition. Ph. D dissertation. MIT, AI-TR 197; 1975, also in book, same title. Elsevier.

TASK	METHODS	SUBTASKS
Design	-Propose, Critique, Modify family (PCM)	-Propose, Verify Critique, Modify
Propose	-Decomposition methods (incl. Design Plans) & Transformation methods	-Specification generation for subproblems
		-Solution of subproblems generated by decomposition (another set of Design-tasks)
		-Composition of subproblem solutions
	-Case-based methods	-Match & retrieve similar case
	-Global constraint-satisfaction methods	
	-Numerical optimization methods	
	-Numerical or Symbolic constraint propagation methods	
Specification generation for subproblems		
	-Constraint propagation incl. constraint posting	-Simulation to decide how constraints propagate
Composition of subproblem solutions	-Configuration methods	-Simulation for prediction behavior of candiate configurations
Verify	-Domain-specific calculations or simulation	
	-Qualitative simulation, Consolidation -Visual simulation	
Critique	-Causal behavioral analysis techniques to assign responsibility	
	-Dependency-analysis techniques	
Modify	-Hill-climbing-like methods which incrementally improve parameters.	
	-Dependency-based changes	
	-Function-to-structure mapping knowledge	
	-Add new functions	-Design new function, Recompose with candidate design

Table 1: For each task, there is a default "Compiled Knowledge" method which has domain-specific knowledge to achieve it directly & which is not included above. For subtasks such as critiquing, we have only indicated families of generic AI methods, without explicit indication of their subtasks.

Corrective Learning by Generating and Comparing Explanations

Michael Weintraub and Tom Bylander

Laboratory for Artificial Intelligence Research
Department of Computer and Information Science
The Ohio State University
Columbus, Ohio 43210
(614) 292-1413

weintrb@cis.ohio-state.edu
byland@cis.ohio-state.edu

Introduction

A common problem with knowledge-based systems is they make mistakes. An important task is to have systems be able to overcome their shortcomings and be able to correct themselves. In knowledge-based systems, representations have concentrated on achieving competence on some task. To this end, many successful systems have been designed and implemented. Even though many systems perform their tasks quite well, these systems can still make mistakes. Errors can result from the system having incomplete or incorrect knowledge. The focus of this work is to identify structures and mechanism to overcome these problems.

The problem of identifying and modifying incorrect knowledge is the theory revision problem. The particular version of the problem being considered is the corrective learning task. In this problem, a system produces an answer in response to the input data that does not match the expert's answer—i.e., it does not match the accepted correct answer—and it must reconcile itself so it will produce the correct answer the next time this situation is encountered. More precisely, given a knowledge-based system, a set of data, the system's answer, and the "correct" answer, the problems are (1) to identify the elements in the system's knowledge structure causing the discrepancy between the two answers and (2) to modify the system such that it will produce the correct answer in future similar situations.

Previous Research

Research in knowledge acquisition has partially addressed this problem by asking a domain expert or using statistics to identify where an error in reasoning occurred and how it might be corrected [6, 13, 9, 2, 1, 18]. For example, Teresias, which serves as a knowledge acquisition tool for Mycin [15], operates by reviewing a trace of a system execution with an expert. Whenever the expert feels the behavior (or the knowledge) of the system is wrong in the context of a given problem, the system tries to elicit knowledge from the expert that will result in the correct behavior. Thus, Teresias relies upon a domain expert to understand the trace explanations from the system, and to identify and modify the incorrect knowledge. The SEEK systems differ because they do not rely upon a domain expert, but instead use statistics of how often a rule was used to get a correct solution as opposed to an incorrect one to identify what might be wrong. Correction occurs by proposing modifications and accepting those which most improve the system's behavior over a set of cases. What SEEK cannot do is produce causal explanations of its corrections in terms of the domain.

Like Teresias, most of these systems rely heavily upon the domain expert to interpret and provide corrections. The domain expert extends the system's knowledge, and the knowledge acquisition task is to try to focus the interaction and produce useful (operational) knowledge to be included into the system. In some sense, the domain expert is playing the role of the domain theory described in explanation-based learning (EBL) [12, 8]. Ideally, a system could learn autonomously if such a theory is represented within the system—provided this theory is complete, consistent, and tractable. Unfortunately, a theory that meets these three criteria is generally not possible. Thus the problem requires that some other information be given. For example in the corrective learning task, the correct answer is also supplied. The purpose of having this additional information is it helps focus the search for incorrect knowledge. In the corrective learning task, the learning mechanism must be able to generate explanations of why the correct answer is right and the system's answer is wrong. Traditional EBL requires that the domain theory have sufficient knowledge to imply the correct answer. But if the domain theory is less than ideal (incomplete, intractable, or inconsistent), the corrective learning task is to construct plausible explanations of why the correct answer is right and the system's answer is wrong, and to have sufficient knowledge to determine the best explanation from those generated.

Credit Assignment as Identifying Subtasks at Fault

The particular problem a system is expected to solve is often quite complex. These tasks, for example: diagnosis, planning, etc., are solved through a

series of subtasks which can be complex in their own right. The system's failure to produce the "correct" answer results from one of the subtasks producing an output that precludes the preferred answer. Each subtask can fail in a variety of ways, and through an analysis of each subtask the possible error types can be specified. Each error type can be thought of as an alternative hypothesis that can potentially explain the system's deviation from the preferred answer. The credit assignment problem is to identify the best set of (performance system) fault hypotheses that explain the deviation from the better answer. Thus the fault identification problem (credit assignment) is abductive in nature. In this abductive task, the data to be explained are the differences between the system's answer and the correct answer and the hypotheses are assertions about incorrect answers made be the system's subtasks [3].

One set of methods that can be used to implement knowledge-based systems are generic tasks [5]. Each generic task is defined by its input, output, and the subtasks used to produce the output from the input. Examples of different generic tasks are hypothesis assembly, classification, and structured matching. Thus for each generic task, there is a defined set of subtasks that can be blamed for causing the incorrect answer. Each one of these subtasks can fail in an enumerable set of ways. By an analysis of how each subtask computes its mapping of inputs to outputs, a set of rules delimiting its possible types of errors can be defined.

If more than one error is applicable in a given situation, the credit assignment system must decide between them and select the error that is the best candidate for explaining the incorrect answer. To make this choice, there must be criteria for evaluating the relative plausibility of each hypothesized error. In our work, this evaluation depends on a partial causal theory of the domain, i.e., instead of a definitive proof, this kind of domain theory provides causal evidence for and against a hypothesis.

Generating Explanations

The key to this problem is the generation of explanations of different types. Explanations can be derived from knowledge about the domain and knowledge about the system. A system might have several types of knowledge. There is control knowledge, associational knowledge, and causal knowledge. Control knowledge defines the problem solving behavior of a system; that is, this knowledge defines the knowledge organization and control mechanisms for searching through this organization. Associational knowledge is derived from experiences, and causal knowledge represents the principles, forces, and components existing in a domain. As mentioned, the power to learn comes from being able to generate plausible explanations. The ability to produce explanations is dependent upon understanding the types of knowledge and

problem-solving representations used within the system. In [16], three types of explanations are identified: trace, control, and justification. An understanding of the system is the key to providing a method for indexing from the system's knowledge and structure into the different types of explanations.

Trace explanations relate how the system performed in the particular case. Control explanations relate the system's actions in terms of its control strategy. Previous work, typified by Teresias, centers around these two types of explanations. In order to automate the learning process, the system needs to be able to generate justification explanations that provide causal domain explanations and how the knowledge used in the system relates to this understanding. Both aspects are necessary. For example in a knowledge-based system, the rule base might not perfectly reflect the underlying domain model because the operationality criterion used to create these rules allowed some features of the domain model to be overgeneralized or omitted.

For an explanation of the knowledge in the system to be generated, an understanding of this knowledge must be represented. One way to do this is to treat the domain as a collection of devices that produce a behavior. Devices have structure, function, and behavior. Structure defines the components and their connections, function defines the use of the device, and behavior is the particular method (states and transitions) used to achieve a function. One representational scheme that captures this notion is the Functional Representation (FR)[14] In an FR, the components, the functions, and the behaviors of a device are represented as a frame. In an FR, enabling conditions and important background assumptions are also encoded. Thus, the FR provides a method for indexing from the problem solving system into the causal domain knowledge. Thus, an explanation of a problem solving behavior can be generated from the underlying knowledge, and the level of the explanation is determined by its use in the system.

Application of These Ideas to QUAWDS

This idea is being applied to QUAWDS, a system that performs human pathologic gait analysis [17]. One goal of gait analysis is to identify the set of muscle faults causing the observed gait motions. QUAWDS uses several different problem solving activities to form its conclusions. Hypothesis assembly [11] is used to synthesize composite diagnostic conclusions from a set of known faults. The subtasks of hypothesis assembly QUAWDS uses are finding generation, hypothesis generation, hypothesis rating, and coverage determination. Each of these subtasks is assigned to the types of problem solving activity that can determine the required information efficiently and effectively. In QUAWDS, the subtasks are determined by different problem solving activities: hierarchical classification [4] and qualitative modeling [17, 10, 7]. By combining the knowledge of how each task is achieved and how it can err with

a causal understanding of the domain, it is possible to determine plausible explanations of why the system produced the incorrect answer.

For example, suppose a patient presents with decreased hip flexion and knee flexion during swing phase.[1] Further suppose QUAWDS proposed overactivity of the hamstrings (the muscle on the back part of the thigh) as an answer, and the domain expert preferred the hypothesis weak hip flexor muscles. From the control explanation, the sequence of steps the system uses to synthesize an answer is determined. The first step in the processing is to identify the candidate (muscle) fault hypotheses. From the trace, it can be determined if the the expert's answer was proposed by the system. If it was not, the fault hypothesis generation subtask is a candidate for having caused the wrong answer. Suppose the system considered the expert's answer, but rejected it because it was not the most plausibly rated hypothesis. The subtask that computes this value is the hypothesis rater. It becomes a candidate for explaining why the system erred.

Two candidate reasons for the incorrect preference of the hypotheses are proposed: the system's answer is overrated and the expert's answer is underrated. One heuristic for selecting between which the alternatives to pursue is to choose the candidate whose fault expectations are best matched in the data. The hamstrings normally operate directly on the hip by extending it and directly on the knee by flexing it. Overactivity causes the expectations increased hip extension (decreased flexion) and increased knee flexion. These expectations are generated by the qualitative model. In this case, the first expectation is matched, but the second is not. Lowering a fault hypothesis' value has the expectation that the hypothesis' expectations should not be met. The hip flexors act directly on the hip to flex it. In swing phase, the knee will passively flex secondary to the amount of hip flexion. Weak muscles generate expectations of decreased motions. Thus the weak hip flexor hypothesis generates the expectations of decreased hip flexion and decreased knee flexion. Both expectations are found in the data. Raising the hypothesis' value has the expectation that the hypothesis' expectations should be met. As the expert solution's expectations are met and the system solution's are not, raising the expert's answer is selected as the best candidate. This process now recurses over the subtasks involved in determining the fault hypothesis' value until an explanation is found for why the weak hip flexor hypothesis is underrated.

Summary

In this paper, we have briefly outlined an abductive model for assigning blame in a problem solver that makes mistakes. This model depends on

[1]Swing phase is the part of the gait cycle when one leg is swinging forward and the other leg is on the ground.

having represented several different types of knowledge so that various types of explanations can be built when needed. By modularizing the problem solver appropriately, the complexity inherent in credit assignment can be managed.

Acknowledgements

This work is supported by National Institute on Disability and Rehabilitation Research grants H133C90090 and H133E80017 and Air Force Office of Scientific Research grant AFOSR 89-0250.

References

[1] R. Bareiss. *Protos: A unified approach to concept representation, classification, and learning.* PhD thesis, Univ. of Texas, 1988.

[2] J. Boose and J. Bradshaw. Expertise transfer and complex problems: using AQUINAS as a knowledge-acquisition workbench for knowledge-based systems. In J. H. Boose and B. R. Gaines, editors, *Knowledge Acquisition Tools for Expert Systems*, volume 2, pages 39–64. Academic Press, New York, 1988.

[3] T. Bylander, D. Allemang, M. C. Tanner, and J. R. Josephsson. The computational complexity of abduction. *Artifical Intelligence (to appear)*, 1984.

[4] T. Bylander and S. Mittal. CSRL: A language for classificatory problem solving and uncertainty handling. *AI Magazine*, pages 66–77, August, 1986.

[5] B. Chandrasekaran. Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design. *IEEE Expert*, 1(3):23–30, 1986.

[6] R. Davis. Applications of meta-level knowledge to the constuction, maintenance, and use of large knowledge bases. In R. Davis and D. Lenat, editors, *Knowledge-Based systems in artificial intelligence*, pages 229–490. McGraw-Hill, New York, 1982.

[7] J. de Kleer. The origin and resolution of ambiguities in causal arguments. In *Proc. Sixth Int. Joint Conf. on Artificial Intelligence*, pages 197–203, Tokyo, 1979.

[8] G. DeJong and R. Mooney. Explanation-based learning: An alternative view. *Machine Learning*, 1:145–176, 1986.

[9] A. Ginsberg, S. Weiss, and P. Politakis. SEEK2:A generalized approach to automatic knowledge base refinement. In *Ninth International Conference on Artifical Intelligence*, pages 367–374, Los Angeles, 1985.

[10] D. Hirsch, S. R. Simon, T. Bylander, M.Weintraub, and P. Szolovits. Using causal reasoning in gait analysis. *Applied Artificial Intelligence*, 3(2-3):253–272, 1989.

[11] J. R. Josephson, B. Chandrasekaran, J. W. Smith, and M. C. Tanner. A mechanism for forming composite explanatory hypotheses. *IEEE Trans. Systems, Man and Cybernetics*, 17(3):445–454, 1987.

[12] T. Mitchell, R. Keller, and S. Kedar-Cebelli. Explanation-based generalization: A unifying view. *Machine Learning*, 1:47–80, 1986.

[13] P. Politakis and S. Weiss. Using empirical analysis to refine expert system knowledge bases. *Artifical Intelligence*, 22(1):23–84, 1984.

[14] V. Sembugamoorthy and B. Chandrasekaran. Functional representation of devices and compilation of diagnostic problem-solving systems. In J. Kolodner and C. Reisbeck, editors, *Experience, Memory, and Reasoning*, pages 47–73. Lawrence Erlbaum Associates, 1988.

[15] E. H. Shortliffe. *Computer-Based Medical Consultations:* MYCIN. Elsevier, New York, 1976.

[16] M. Tanner. *Explaining Knowledge Systems: Jutifying Diagnostic Conclusions.* PhD thesis, Ohio State University, 1989.

[17] M. Weintraub and T. Bylander. QUAWDS: A composite diagnostic system for gait analysis. *Computer Methods and Programs in Biomedicine (to appear)*, 1990.

[18] D. Wilkens and K. Tan. Knowledge base refinement as improving an incorrect, inconsistent and incomplete domain theory. In *The Sixth International Workshop on Machine Learning*, pages 332–337, Ithaca, N.Y., 1989.

The Computational Complexity of Abduction

Tom Bylander, Dean Allemang,
Michael C. Tanner, and John R. Josephson
Laboratory for Artificial Intelligence Research
Department of Computer and Information Science
The Ohio State University
Columbus, Ohio

June 20, 1990

to appear in *Artificial Intelligence*

Abstract

The problem of abduction can be characterized as finding the best explanation of a set of data. In this paper we focus on one type of abduction in which the best explanation is the most plausible combination of hypotheses that explains all the data. We then present several computational complexity results demonstrating that this type of abduction is intractable (NP-hard) in general. In particular, choosing between incompatible hypotheses, reasoning about cancellation effects among hypotheses, and satisfying the maximum plausibility requirement are major factors leading to intractability. We also identify a tractable, but restricted, class of abduction problems.

---

*Thanks to B. Chandrasekaran, Ashok Goel, Jack Smith, and Jon Sticklen for their comments on the numerous versions of this paper. The referees have also made a substantial contribution. Any remaining errors are our responsibility, of course. This research has been supported in part by the National Library of Medicine, grant LM-04298; the National Heart, Lung and Blood Institute, NIH Grant 1 R01 HL 38776-01; the National Science Foundation through a graduate fellowship; the Defense Advanced Research Projects Agency, RADC contract F30602-85-C-0010; the Air Force Office of Scientific Research, contract F49620-89-C-0110; and the National Science Foundation, grant IRI-8902142.

†This paper is a revised and integrated version of Allemang et al. [1] and Bylander et al. [2].

‡Dean Allemang's current address is Institut für Informatik, Universität Zürich, Winterthurerstrasse 190, CH-8057 Zürich, Switzerland.

1 Introduction

The problem of abduction can be characterized as finding the best explanation of a set of data [13]. Abduction applies to a wide variety of reasoning tasks [3]. For example, in medical diagnosis, the final diagnosis explains the signs and symptoms of the patient [23, 25]. In natural language understanding, the intended interpretation of a sentence explains why the sentence was said [12]. In scientific theory formation, the acceptance of a hypothesis is based on how well it explains the evidence [31].

What kinds of abduction problems can be solved efficiently? To answer this question, we must formalize the problem and then consider its computational complexity. However, it is not possible to prescribe a specific complexity threshold for all abduction problems. If the problem is "small," then exponential time might be fast enough. If the problem is sufficiently large, then even $O(n^2)$ might be too slow. However, for the purposes of analysis, the traditional threshold of intractability, NP-hard, provides a rough measure of what problems are impractical [10]. Clearly, NP-hard problems will not scale up to larger, more complex domains.

Our approach is the following. First, we formally characterize abduction as a problem of finding the most plausible composite hypothesis that explains all the data. Then we consider several classes of problems of this type, the classes being differentiated by additional constraints on how hypotheses interact. We demonstrate that the time complexity of each class is polynomial (tractable) or NP-hard (intractable), relative to the complexity of computing the plausibility of hypotheses and the data explained by hypotheses.

Our results show that this type of abduction faces several obstacles. Choosing between incompatible hypotheses, reasoning about cancellation effects among hypotheses, and satisfying the maximum plausibility requirement are major factors making abduction intractable in general.

Some restricted classes of abduction problems are tractable. One kind of class is when some constraint guarantees a polynomial search space, e.g., the single-fault assumption (more generally, a limit on the size of composite hypotheses), or if all but a small number of hypotheses can be ruled out.[1] This kind of class trivializes complexity analysis because exhaustive search over the possible composite hypotheses becomes a tractable strategy.

However, we have discovered one class of abduction problems in which hypothesis assembly [13] can find the best explanation without exhaustive search. Informally, the constraints that define this class are: no incompatibility relationships, no cancellation interactions, the plausibilities of the individual hypotheses are all different from each other, and one explanation is qualitatively better than any other explanation. Unfortunately, it is intractable to determine whether the last condition holds. We consider one abduction system in which hypothesis assembly was applied, so as to examine the ramifications of these constraints in a real world situation.

The remainder of this paper is organized as follows. First, we provide a brief historical background to abduction. Then, we define our model of abduction problems and show how it applies to other theories of abduction. Next, we describe our complexity results, proofs

[1] The latter constraint is not the same as "eliminating candidates" in de Kleer & Williams [6] or "inconsistency" in Reiter [26]. If a hypothesis is insufficient to explain all the observations, the hypothesis is not ruled out because it can still be in composite hypotheses.

of which are given in the appendix. Finally, we consider the relationship of these results to one abduction system.

2 Background

C. S. Peirce, who first described abductive inference [20], provided two intuitive characterizations: given an observation d and the knowledge that h causes d, it is an abduction to hypothesize that h occurred; and given a proposition q and the knowledge $p \rightarrow q$, it is an abduction to conclude p. In either case, an abduction is uncertain because something else might be the actual cause of d, or because the reasoning pattern is the classical fallacy of "affirming the consequent" and is formally invalid. Additional difficulties can exist because h might not always cause d, or because p might imply q only by default. In any case, we shall say that h *explains* d and p *explains* q, and we shall refer to h and p as *hypotheses* and d and q as *data*.

Pople pointed out the importance of abduction to AI [23], and he with Miller and Myers implemented one of the earliest abduction systems, INTERNIST-I, which performed medical diagnosis in the domain of internal medicine [16, 24]. This program contained an explicit list of diseases and symptoms, explicit causal links between the diseases and the symptoms, and probabilistic information associated with the links. INTERNIST-I used a form of hill climbing—once a disease outscored its competitors by a certain threshold, it was permanently selected as part of the final diagnosis. Hypothesis assembly [13] is a generalization of this technique. Below, we describe a restricted class of problems for which hypothesis assembly can efficiently find the best explanation.

Based on similar explicit representations, Pearl [19] and Peng & Reggia [21] find the most probable composite hypothesis that explains all the data, a task that is known to be intractable in general [4]. Below we describe additional constraints under which this task remains intractable.

In contrast to maintaining explicit links between hypotheses and data, Davis & Hamscher's model-based diagnosis [5] determines at run-time what data need to be explained and what hypotheses can explain the data. Much of this work, such as de Kleer & Williams [6] and Reiter [26], place an emphasis on generating all "minimal" composite hypotheses that explain all the data. However, there can be an exponential number of such hypotheses. Current research is investigating how to focus the reasoning on the most relevant composite hypotheses [7, 8, 30]. However, we show below that it is intractable in general to find a composite hypothesis that explains all the data, and that even if it is easy to find explanations, generating all the relevant composite hypotheses is still intractable.

Whatever the technique or formulation, certain fundamentals of the abduction task do not change. In particular, our analysis shows how computational complexity arises from constraints on the explanatory relationship from hypotheses to data and on plausibility ordering among hypotheses. These constraints do not depend on the style of the representation or reasoning method (causal vs. logical, probabilistic vs. default, explicit vs. model-based, ATMS or not, etc.). In other words, certain kinds of abduction problems are hard no matter what representation or reasoning method is chosen.

3 Notation, Definitions, and Assumptions

We use the following notational conventions and definitions. d stands for a datum, e.g., a symptom. D stands for a set of data. h stands for an individual hypothesis, e.g., a hypothesized disease. H stands for a set of individual hypotheses. H can be treated as a composite hypothesis, i.e., each $h \in H$ is hypothesized to be present, and each $h \notin H$ is hypothesized to be absent or irrelevant.

3.1 Model of Abduction

An *abduction problem* is a tuple $\langle D_{all}, H_{all}, e, pl \rangle$, where:

D_{all} is a finite set of all the data to be explained,

H_{all} is a finite set of all the individual hypotheses,

e is a map from subsets of H_{all} to subsets of D_{all} (H explains $e(H)$), and

pl is a map from subsets of H_{all} to a partially ordered set (H has plausibility $pl(H)$)

For the purpose of this definition and the results below, it does not matter whether $pl(H)$ is a probability, a measure of belief, a fuzzy value, a degree of fit, or a symbolic likelihood. The only requirement is that the range of pl is partially ordered.

H is *complete* if $e(H) = D_{all}$. That is, H explains all the data.

H is *parsimonious* if $\nexists H' \subset H\ (e(H) \subseteq e(H'))$. That is, no proper subset of H explains all the data that H does.

H is an *explanation* if it is complete and parsimonious. That is, H explains all the data and has no explanatorily superfluous elements. Note that an explanation exists if and only if a complete composite hypothesis exists.[2]

H is a *best explanation* if it is an explanation, and if there is no explanation H' such that $pl(H') > pl(H)$. That is, no other explanation is more plausible than H. It is just "a best" because pl might not impose a total ordering over composite hypotheses (e.g., because of probability intervals or qualitative likelihoods). Consequently, several composite hypotheses might satisfy this definition.

3.2 Relation to Other Work

These definitions are intended to formalize the notion of best explanation in Josephson et al. [13]. However, our definitions are not limited to that paper. We consider in detail here how Reiter's theory of diagnosis [26] and Pearl's theory of belief revision [19] can be mapped to our model of abduction.

[2]Composite hypotheses that do not explain all the data can still be considered explanations, albeit partial. Nevertheless, because explaining all the data is a goal of the abduction problems we are considering, for convenience, this goal is incorporated into the definition of "explanation."

3.2.1 Reiter's Theory of Diagnosis

Reiter defines a diagnosis problem as a tuple (SD, COMPONENTS, OBS), in which SD and OBS are finite sets of first-order sentences comprising the system description and observations, respectively. COMPONENTS is a finite set of constants; and AB is a distinguished unary predicate, interpreted as abnormal. A diagnosis is defined to be a minimal set $\Delta \subseteq$ COMPONENTS such that:

$$\text{SD} \cup \text{OBS} \cup \{\text{AB}(c) \mid c \in \Delta\} \cup \{\neg\text{AB}(c) \mid c \in \text{COMPONENTS} \setminus \Delta\}$$

is consistent. "Minimal set" means that no subset of Δ satisfies the same condition.

Each subset of COMPONENTS can be treated as a composite hypothesis, i.e., a conjecture that certain components are abnormal, and that all other components are normal. A diagnosis problem can then be mapped into an abduction problem as follows:

$$\begin{aligned} H_{all} &= \text{COMPONENTS} \\ D_{all} &= \text{OBS} \\ e(H) &= \text{a maximal set } D \subseteq D_{all} \text{ such that}^3 \\ &\quad \text{SD} \cup D \cup \{\text{AB}(h) \mid h \in H\} \cup \{\neg\text{AB}(h) \mid h \in H_{all} \setminus H\} \\ &\quad \text{is consistent.} \end{aligned}$$

A solution for the diagnosis problem then corresponds to an explanation for the abduction problem, and vice versa. Reiter does not define any criteria for ranking diagnoses, so there is nothing to map to *pl*.

3.2.2 Pearl's Theory of Belief Revision

A Bayesian belief network [18] is a directed acyclic graph whose nodes $\boldsymbol{W}$ are propositional variables. The probabilistic dependencies between the variables are described by specifying $P(x|s)$ for each value assignment x to a variable $X \in \boldsymbol{W}$ and each value assignment s to X's parents S.[4] The intention is that "the arcs signify the existence of direct causal influences between the linked propositions, and the strengths of the these influences are quantified by the conditional probabilities of each variable given the state of its parents" [19, p. 175].

For a particular belief revision problem [19], some subset $\boldsymbol{V}$ of the variables $\boldsymbol{W}$ are initialized with specific values. Let $\boldsymbol{v}$ be the value assignment to $\boldsymbol{V}$. The solution to the problem is the most probable value assignment $\boldsymbol{w}^*$ to all the variables $\boldsymbol{W}$, i.e., $P(\boldsymbol{w}^*|\boldsymbol{v})$ is greater than or equal to $P(\boldsymbol{w}|\boldsymbol{v})$ for any other value assignment $\boldsymbol{w}$ to the variables $\boldsymbol{W}$. $\boldsymbol{w}^*$ is called the most probable explanation (MPE).

$\boldsymbol{v}$ can be mapped to the set of data to be explained, i.e., a value assignment x to a variable $X \in \boldsymbol{V}$ is a datum. $\boldsymbol{v}$ can be explained by appropriate value assignments to the other variables $\boldsymbol{W} \setminus \boldsymbol{V}$. Treating value assignments of *true* as individual hypotheses, a belief revision problem can be be mapped to an abduction problem as follows:

[3] There might be more than one maximal subset of observations that satisfies these conditions. If so, then $e(H)$ selects some preferred subset.

[4] For belief networks, we use a (boldface) lower case letter to stand for a (set of) value assignment(s) to a (set of) variable(s), which is denoted by a (boldface) upper case letter.

$$
\begin{aligned}
D_{all} &= \boldsymbol{v} \\
H_{all} &= \boldsymbol{W} \setminus \boldsymbol{V} \\
e(H) &= \text{a maximal set } D \subseteq D_{all} \text{ such that} \\
&\qquad P(H = true \wedge H_{all} \setminus H = false | D) > 0 \\
pl(H) &= P(H = true \wedge H_{all} \setminus H = false | e(H))
\end{aligned}
$$

The MPE corresponds to a complete composite hypothesis. If the MPE is also parsimonious, then it corresponds to the best explanation.[5] However, the MPE might assign *true* to more variables than necessary for explanatory purposes. In the context of other value assignments, $X = true$ might be more likely than $X = false$ even if $X = true$ is superfluous under the above mapping [21].

This lack of correspondence between the MPE and the best explanation can be rectified by creating, for each $X \in \boldsymbol{W} \setminus \boldsymbol{V}$, a dummy variable X' and a dummy value assignment that can be "caused" only if $X \neq X'$. With this modification, the MPE corresponds to the best explanation.

Another way of rectifying the situation is to simply ignore the parsimony constraint. With this in mind, we shall use the mapping given above.

3.2.3 Other Theories of Abduction

These reductions from problems in Reiter's and Pearl's theories to abduction problems provide strong evidence that our model of abduction is general enough to accommodate any theory of abduction, e.g., [6, 15, 21, 22]. This is because our model leaves e and pl virtually unconstrained. We exploit this freedom below by defining and analyzing natural constraints on e and pl without considering the representations—logical, causal, or probabilistic—underlying the computation of e and pl. To make the analysis complete, we also show how some of these constraints can be reduced to problems in Reiter's and Pearl's theories.

3.3 Tractability Assumptions

In our complexity analysis, we assume that e and pl are tractable. We also assume that e and pl can be represented reasonably, in particular, that the size of their internal representations is polynomial in $|D_{all}| + |H_{all}|$.

Clearly, the tractability of these functions is central to abduction, since it is difficult to find plausible hypotheses explaining the data if it is difficult to compute e and pl. This should not be taken to imply that the tractability of these functions can be taken for granted. For example, it can be intractable to determine explanatory coverage of a composite hypothesis [26] and to calculate the probability that an individual hypothesis is present, ignoring other hypotheses [4]. We make these assumptions to simplify our analysis of abduction problems. To reflect the complexity of these functions in our tractability results, we denote the time complexity of e and pl with respect to the size of an abduction problem as C_e and C_{pl}, respectively, e.g., nC_e indicates n calls to e.

[5] One difficulty with the more "natural" mapping $pl(H) = P(H = true | \boldsymbol{v})$ is that even if the MPE is parsimonious, it might not be the best explanation.

For convenience, we assume the existence and the tractability of a function that determines which individual hypotheses can contribute to explaining a datum. Although it is not a true inverse, we refer to this function as e^{-1}, formally defined as:

$$e^{-1}(d) = \{h \mid \exists H \subset H_{all}\ (d \notin e(H) \wedge d \in e(H \cup \{h\}))\}$$

Note that $h \in e^{-1}(d)$ does not imply $d \in e(h)$.

The key factors, then, that we consider in the complexity of finding a best explanation are properties of e and pl that allow or prevent tractable computation given that e, e^{-1}, and pl can be computed "easily." That is, given a particular class of abduction problems, how much of the space of composite hypotheses must be explicitly searched to find a best explanation? As demonstrated below, intractability is the usual result in classes of problems that involve significant interaction among the elements of composite hypotheses.

3.4 Simplifications

We should note that these definitions and assumptions simplify several aspects of abduction. For example, we define composite hypotheses as simple combinations of individual hypotheses. In reality, the relationships among the parts of an abductive answer and the data being explained can be much more complex, both logically and causally.

Another simplification is that *domains* are not defined. One way to do this would be to specify what data are possible (D_{poss}) and general functions for computing explanatory coverage and plausibilities based on the data (e_{gen} and pl_{gen}). Then for a specific abduction problem, the following constraints would hold: $D_{all} \subseteq D_{poss}$, $e(H) = e_{gen}(H, D_{all})$, and $pl(H) = pl_{gen}(H, D_{all})$ (cf. Allemang et al. [1]).

The definitions of abduction problems or domains do not mention the data that do not have to explained, even though they could be important for determining e and pl. For example, the age of a patient does not have to be explained, but can influence the plausibility of a disease. We shall assume that e and pl implicitly take into account data that do not have to be explained, e.g., in the definition of domains above, these data can be an additional argument to e_{gen} and pl_{gen}.

Despite these simplifications, our analysis provides powerful insights concerning the computational complexity of abduction.

3.5 An Example

We shall use the following example to facilitate our discussion:

$$\begin{aligned}
H_{all} &= \{h_1, h_2, h_3, h_4, h_5\} \\
D_{all} &= \{d_1, d_2, d_3, d_4\} \\
e(h_1) &= \{d_1\} & pl(h_1) &= \text{superior} \\
e(h_2) &= \{d_1, d_2\} & pl(h_2) &= \text{excellent} \\
e(h_3) &= \{d_2, d_3\} & pl(h_3) &= \text{good} \\
e(h_4) &= \{d_2, d_4\} & pl(h_4) &= \text{fair} \\
e(h_5) &= \{d_3, d_4\} & pl(h_5) &= \text{poor}
\end{aligned}$$

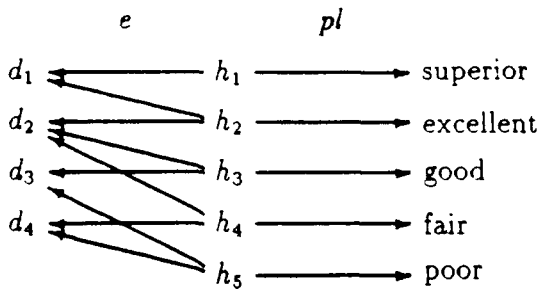


Figure 1: Example of an Abduction Problem

Figure 1 is a pictorial representation of the example. The values of *pl* should simply be interpreted as indicating relative order of plausibility. If $e(H)$ is the union of $e(h)$ for $h \in H$, then $\{h_2, h_3, h_5\}$ is complete, but not parsimonious since h_3 is superfluous. $\{h_2, h_3\}$ is parsimonious, but not complete since it does not explain d_4. Based on the plausibility ordering criterion defined in Section 5, $\{h_1, h_3, h_4\}$ and $\{h_2, h_5\}$ would be considered the best explanations.

Using these definitions, assumptions, and example, we first discuss how properties of *e* affect the tractability of finding best explanations, and then consider properties of *pl*.

4 Complexity of Finding Explanations

4.1 Independent Abduction Problems

In the simplest problems, an individual hypothesis explains a specific set of data regardless of what other individual hypotheses are being considered. This constraint is assumed by INTERNIST-I [16], Reggia's set covering algorithm [25], Peng & Reggia's parsimonious covering theory [21], Pearl's belief revision theory if interactions are restricted to noisy-OR (an effect can occur only if one or more of its causes are present) [19], and Eshelman's cover-and-differentiate method [9]. The use of conflict sets [6, 26] also appears to make this assumption—each conflict set corresponds to a datum to be explained, and the elements of the conflict set correspond to the hypotheses that can independently explain the datum.

Formally, an abduction problem is *independent* if:

$$\forall H \subseteq H_{all}\ (e(H) = \bigcup_{h \in H} e(h))$$

That is, a composite hypothesis explains a datum if and only if one of its elements explains the datum. This constraint makes explanatory coverage equivalent to set covering [25]. Assuming independence, the explanations in our example (refer to Figure 1) are: $\{h_1, h_3, h_4\}$, $\{h_1, h_3, h_5\}$, $\{h_1, h_4, h_5\}$, $\{h_2, h_3, h_4\}$, and $\{h_2, h_5\}$.

One way to find a best explanation would be to generate all explanations and then sort them by plausibility. However, it is well-known that there can be an exponential number of explanations. It is not surprising then that determining the number of explanations is hard

W stands for the working composite hypothesis.
Nil is returned if no explanation exists.

Determine whether an explanation exists.
If $e(H_{all}) \neq D_{all}$ then
    Return nil

Find an explanation.
$W \leftarrow H_{all}$
For each $h \in H_{all}$
    If $e(W \backslash \{h\}) = D_{all}$ then
        $W \leftarrow W \backslash \{h\}$
Return W

Algorithm 1: Finding an Explanation in Independent and Monotonic Abduction Problems

Theorem 4.1 *For the class of independent abduction problems, it is #P-complete to determine the number of explanations.*

That is, determining the number of explanations for an independent abduction problem is just as hard as determining the number of solutions to an NP-complete problem.[6]

The definition of best explanation, however, does not require that all explanations be explicitly enumerated. For example, if h is the most plausible individual hypothesis, h explains D_{all}, and $H \subset H'$ implies $pl(H) > pl(H')$, then h can be declared to be the best explanation without further search. In general, the task of finding a best explanation can be divided into two subtasks: (1) find one explanation, and (2) repeatedly find better explanations until a best one is found. In the remainder of this section then, we shall consider the complexity of generating one or more explanations. The following section discusses the complexity of finding better explanations.

For independent abduction problems, it is tractable to find an explanation. Let $n = |D_{all}| + |H_{all}|$.

Theorem 4.2 *For the class of independent abduction problems, there is an $O(nC_e + n^2)$ algorithm for finding an explanation, if one exists.*

Algorithm 1 performs this task within this order of complexity. The appendix gives a detailed explanation of this algorithm, but we note several aspects of its operation here.

It is easy to check whether an explanation exists. If $\bigcup_{h \in H_{all}} e(h) \neq D_{all}$, then a union over any subset of H_{all} will not equal D_{all} either.

The loop makes one pass through the individual hypotheses. It examines each individual hypothesis in turn and removes it if no explanatory coverage is lost. Only one pass is

[6] Also, it is #P-complete to determine the number of complete composite hypotheses. The definition of #P-complete comes from Valiant [32].
Detailed proofs of Theorem 4.1 and other theorems are given in the appendix.

necessary because if the result W had a superfluous element h, then h would have been superfluous for any superset of W, and thus, would have been removed by the body of the loop.

If the e^{-1} function is available (see Section 3.3 for the definition of e^{-1}), then the working hypothesis W, instead of being initialized to H_{all}, can be initialized to include only one element from $e^{-1}(d)$ for each $d \in D_{all}$. This modification has an advantage if e^{-1} is easy to compute and the working hypothesis remains "small."

4.2 Monotonic Abduction Problems

We now consider a more general kind of problem, in which a composite hypothesis can explain additional data that are not explained by any of its elements. For example, suppose the two inputs to an AND-gate are supposed to be 0, and so the output is supposed to be 0, but the observed output of the AND-gate is 1. If the inputs are produced by components A and B, then hypothesizing a single fault in A or B is insufficient to account for the datum, but faults in both A and B are sufficient.

This sort of interaction can also occur if two individual hypotheses ha e an additive interaction. For example, each of the two hypotheses can explain a small value of some measurement, but together can explain a larger measurement. In this latter case, if h only partially explains d, then $d \notin e(h)$. Note though that if adding h to a composite hypothesis can result in completely explaining d, then $h \in e^{-1}(d)$.

Formally, an abduction problem is *monotonic* [1] if:

$$\forall H, H' \subseteq H_{all}\ (H \subseteq H' \rightarrow e(H) \subseteq e(H'))$$

That is, a composite hypothesis does not "lose" any data explained by any of its subsets and might explain additional data. All independent abduction problems are monotonic, but a monotonic abduction problem is not necessarily independent. If, in Figure 1, $\{h_2, h_3\}$ also explained d_4, then $\{h_2, h_3\}$ would also be an explanation and $\{h_2, h_3, h_4\}$ would not be. Monotonic abduction problems from the literature include Josephson's hypothesis assembly technique [1] and Pearl's belief revision theory if interactions are restricted to noisy-OR and noisy-AND [19].

Because the class of monotonic abduction problems includes the independent class, it is also hard to determine the number of explanations. In addition, we have shown that it is hard to enumerate a polynomial number of explanations.

Theorem 4.3 *For the class of monotonic abduction problems, given a set of explanations, it is NP-complete to determine whether an additional explanation exists.*

We have proven this result by a reduction from the class of independent incompatibility abduction problems, which is described below. The idea of the reduction is that the addition of an individual hypothesis to a composite hypothesis can explain the rest of the data, make nearly all the elements of the composite hypothesis superfluous, and result in a previously generated explanation. It turns out to be difficult to generate an additional explanation while avoiding this kind of interaction. Whether a similar result holds for independent abduction problems is an open question.

Although the class of monotonic problems is a superset of the class of independent problems, it is just as efficient to find an explanation. Again, let $n = |D_{all}| + |H_{all}|$.

Theorem 4.4 *For the class of monotonic abduction problems, there is an $O(nC_e + n^2)$ algorithm for finding an explanation, if one exists.*

Algorithm 1 performs this task within this order of complexity. Because of the monotonicity constraint, H_{all} must explain as much or more data than any other composite hypothesis. The loop in Algorithm 1 works for the same reasons as for independent abduction problems. Also, it is possible to use e^{-1} to initialize W, though one must be careful because more than one element from $e^{-1}(d)$ might be needed to explain d.

4.3 Incompatibility Abduction Problems

Implicit in the formal model so far is the assumption that any collection of individual hypotheses is possible. However, most domains have restrictions that invalidate this assumption. For example, a faulty digital switch cannot simultaneously be stuck-at-1 and stuck-at-0. More generally, the negation of a hypothesis can also be considered a hypothesis.

This kind of problem is neither independent nor monotonic because any composite hypothesis that contains a pair of mutually exclusive hypotheses cannot be an acceptable hypothesis, while a subset that excludes at least one hypothesis from each pair is acceptable. We call this kind of problem an *incompatibility abduction problem.*

Formally, an incompatibility abduction problem is a tuple $\langle D_{all}, H_{all}, e, pl, \mathcal{I} \rangle$, where D_{all}, H_{all}, e, and pl are the same as before and $\mathcal{I}$ is a set of two-element subsets of H_{all}, indicating pairs of hypotheses that are incompatible with each other.[7] For an incompatibility problem:

$$\forall H \subseteq H_{all} \left((\exists I \in \mathcal{I} \, (I \subseteq H)) \rightarrow e(H) = \emptyset \right)$$

By this formal trick, a composite hypothesis containing incompatible hypotheses explains nothing, preventing such a composite from being complete (except for trivial cases) or a best explanation.

An *independent incompatibility abduction problem* satisfies the formula:

$$\forall H \subseteq H_{all} \left((\nexists I \in \mathcal{I} \, (I \subseteq H)) \rightarrow e(H) = \bigcup_{h \in H} e(h) \right)$$

That is, except for incompatibilities, the problem is independent. In Figure 1, if $\mathcal{I} = \{\{h_1, h_2\}, \{h_2, h_3\}, \{h_3, h_4\}, \{h_4, h_5\}\}$, then only $\{h_1, h_3, h_5\}$ and $\{h_2, h_5\}$ would be explanations.

Incompatibility abduction problems are more complex than monotonic or independent abduction problems:

Theorem 4.5 *For the class of independent incompatibility abduction problems, it is NP-complete to determine whether an explanation exists.*

[7] Incompatible pairs are the most natural case, e.g., one hypothesis of the pair is the negation of the other. n mutually exclusive hypotheses can be represented as $n(n-1)/2$ incompatible pairs. Incompatible triplets (any two of the three, but not all three) and so on are conceivable, but allowing these possibilities in the formal definition do not affect the complexity results.

We have proven this result by reduction from 3SAT [10], which is satisfiability of boolean expressions in conjunctive normal form, with no more than three literals in any conjunct. Informally, the reduction works as follows. Each 3SAT literal and its negation corresponds to an incompatible pair of hypotheses. Each conjunct of the boolean expression corresponds to a datum to be explained. Satisfying a conjunct corresponds to a hypothesis explaining a datum. Clearly then, a complete composite hypothesis exists iff the boolean expression is satisfiable. Furthermore, a complete composite hypothesis exists iff an explanation exists. Our proof shows that only $O(|H_{all}|)$ incompatible pairs are needed to give rise to intractability.

The underlying difficulty is that the choice between a pair of incompatible hypotheses cannot be made locally, but is dependent on the choices from all other incompatible pairs. It is interesting to note the parsimony constraint plays no role in this result. Just finding a complete composite hypothesis is hard in incompatibility abduction problems.

It follows that:

Corollary 4.6 *For the class of independent incompatibility abduction problems, it is NP-hard to find a best explanation.*

The class of incompatibility abduction problems can be reduced to both Reiter's theory of diagnosis [26] and Pearl's theory of belief revision [19].

Theorem 4.7 *For the class of diagnosis problems, relative to the complexity of determining whether a composite hypothesis is consistent with* SDUOBS, *it is NP-complete to determine whether a diagnosis exists.*

For this theorem, a composite hypothesis is a conjecture that certain components are abnormal, and that all other components are normal.

It is easy to translate the explanatory interactions of an independent incompatibility abduction problem into first-order sentences. For example, $e^{-1}(d) = H$ can be translated to $\text{MANIFEST}(d) \rightarrow \bigvee_{h \in H} \text{AB}(h)$. $\{h, h'\} \in \mathcal{I}$ can be translated to $\text{AB}(h) \rightarrow \neg\text{AB}(h')$. It is interesting that this problem is hard even if it is easy to determine the consistency of a composite hypothesis.

Theorem 4.8 *For the class of belief revision problems, it is NP-complete to determine whether there is a value assignment $\boldsymbol{w}$ to the variables $\boldsymbol{W}$ such that $P(\boldsymbol{w}|\boldsymbol{v}) > 0$.*

This theorem directly follows from Cooper's result that it is NP-complete to determine whether $P(X = true) > 0$ for a given variable X within a belief network [4]. Also, a reduction from incompatibility abduction problems can be done as follows. Map each $h \in H_{all}$ to a "hypothesis" variable. Map each $d \in D_{all}$ to a "data" variable that can be true only if one or more of the hypothesis variables corresponding to $e^{-1}(d)$ are true (e.g., noisy-OR interaction). Map each incompatible pair into a data variable that can be true only if at most one, but not both, of the two corresponding hypothesis variables is true (e.g., NAND). Initializing all the data variables to *true* sets up the problem.

4.4 Cancellation Abduction Problems

Another interaction not allowed in independent or monotonic abduction problems is cancellation, i.e., when one element of a composite hypothesis "cancels" a datum that another element would otherwise explain. Cancellation can occur when one hypothesis can have a subtractive effect on another. This is common in medicine, e.g., in the domain of acid-base disorders, one disease might explain an increased blood pH, and another might explain a decreased pH, but together the result might be a normal pH [17]. Different faults in different components can result in cancellation, e.g., a stuck-at-1 input into an AND-gate might account for an output of 1, but not if the other input is stuck-at-0. Cancellation commonly occurs in the physical world. Newton's second law implies that forces can cancel each other. Cancellation in the form of feedback control is intentionally designed into devices.

Formally, we define a *cancellation abduction problem* as a tuple $\langle D_{all}, H_{all}, e, pl, e_+, e_- \rangle$. e_+ is a map from H_{all} to subsets of D_{all} indicating what data each hypothesis "produces." e_- is another map from H_{all} to subsets of D_{all} indicating what data each hypothesis "consumes." $d \in e(H)$ iff the number of hypotheses in H that produce d outnumber the hypotheses that consume d. That is:

$$d \in e(H) \leftrightarrow |\{h \mid h \in H \wedge d \in e_+(h)\}| > |\{h \mid h \in H \wedge d \in e_-(h)\}|$$

In Figure 1, if we let $e_+ = e$ for individual hypotheses and if $e_-(h_1) = \{d_3\}$, $e_-(h_2) = \{d_4\}$, and $e_-(h_3) = e_-(h_4) = e_-(h_5) = \emptyset$, then the only explanations would be $\{h_1, h_3, h_5\}$ and $\{h_2, h_4, h_5\}$.

Admittedly, this is a simplified model of cancellation effects, in the sense that it captures only one kind of cancellation interaction. Nevertheless, it is sufficient to derive intractability:

Theorem 4.9 *For the class of cancellation abduction problems, it is NP-complete to determine whether an explanation exists.*

We have proven this by reduction from finding explanations in incompatibility abduction problems. Informally, the idea of the reduction is based on the following. Suppose that a datum has two potential "producers" and two potential "consumers." Now any composite hypothesis that contains both consumers cannot explain the datum. In effect, the two consumers are incompatible. Our reduction ensures that each incompatible pair in the incompatibility abduction problem is appropriately mapped to such a situation in the corresponding cancellation abduction problem. Only $O(|H_{all}|)$ "cancellations" are needed for this result, where $\sum_{h \in H_{all}} |e_-(h)|$ gives the number of cancellations.

It follows that:

Corollary 4.10 *For the class of cancellation abduction problems, it is NP-hard to find a best explanation.*

One aspect of cancellation abduction problems is more complex than incompatibility abduction problems. In an independent incompatibility abduction problem, if a complete composite hypothesis is found, then it is easy to find a parsimonious subset. However, this is not true for cancellation abduction problems.

Theorem 4.11 *For the class of cancellation abduction problems, it is coNP-complete to determine whether a complete composite hypothesis is parsimonious.*

Table 1: Computational Complexity of Finding Explanations

class of problems	condition to achieve		
	finding all explanations	finding an explanation	finding a best explanation
independent	NP	P	?
monotonic	NP	P	?
incompatibility	NP	NP	NP
cancellation	NP	NP	NP

P = known polynomial algorithm NP = NP-hard

That is, it is NP-complete to determine whether a complete composite hypothesis is not parsimonious. The idea of our reduction is the following. If a datum has three "producers" and two "consumers," we can ensure that the datum is explained by including all three producers in the composite hypothesis. However, there might be a more parsimonious composite hypothesis in which some of the producers are omitted, but finding such a composite hypothesis means that one or both consumers must be omitted as well, making them effectively incompatible.

Table 1 summarizes the results of this section. The "?" indicates that we not have yet described the complexity of finding a best explanation in independent and monotonic abduction problems.

5 Complexity of Plausibility

To analyze the complexity of finding a best explanation, we need to define how to compare the plausibilities of explanations. We consider one plausibility criterion based on comparing the plausibilities of the elements of the explanations. Other plausibility criteria are considered in Bylander et al. [2], but they are less relevant to other theories of abduction.

5.1 The Best-Small Plausibility Criterion

Everything else being equal, smaller explanations are preferable to larger ones, and more plausible individual hypotheses are preferable to less plausible ones. Thus, in the absence of other information, it is reasonable to compare the plausibility of explanations based on their sizes and the relative plausibilities of their elements. When a conflict occurs, e.g., one explanation is smaller, but has less plausible elements, no ordering can be imposed without additional information.

The *best-small* plausibility criterion formally characterizes these considerations as follows:

$$
\begin{aligned}
&pl(H) > pl(H') \quad \leftrightarrow \\
&\exists m\colon H \rightarrow H' \,(m \text{ is 1-1} \wedge \\
&\qquad \forall h \in H \,(pl(h) \geq pl(m(h))) \wedge \\
&\qquad (|H| = |H'| \rightarrow \exists h \in H \,(pl(h) > pl(m(h)))))
\end{aligned}
$$

That is, to be more plausible according to best-small, the elements of H need to be matched to the elements of H' so that the elements of H are at least as plausible as their matches in H'. If H and H' are the same size, then in addition some element in H must be more plausible than its match in H'. Note that if H is larger than H', then $pl(H) \not> pl(H')$. In Figure 1, $\{h_1, h_3, h_4\}$ and $\{h_2, h_5\}$ would be the best explanations.

We have demonstrated that it is intractable to find best explanations using best-small.

Theorem 5.1 *For the class of independent abduction problems using the best-small plausibility criterion, it is NP-hard to find a best explanation.*

The simplest proof of this theorem involves a reduction from minimum cover [10]. If each individual hypothesis is given the same plausibility, then the smallest explanations (the covers with the smallest sizes) are the best explanations. A more general proof is a reduction from a special class of independent incompatibility abduction problems in which each individual hypothesis is in exactly one incompatible pair. In this reduction, each incompatible pair is mapped into two equally plausible hypotheses, at least one of which must be chosen. If the incompatibility abduction problem has any explanations, they turn out to be best explanations in the best-small problem.

We conjecture that it is possible to reduce from finding a best explanation using best-small to finding a best explanation using any "theory of belief" in which composite hypotheses that are smaller or have more plausible elements can have higher belief values. Of course, standard probability theory is an example of such a theory, as are all its main competitors. This conjecture is supported by the following theorem.

Theorem 5.2 *For the class of belief revision problems restricted to OR interactions, it is NP-hard to find the MPE.*

The restriction to OR interactions means that each effect can be true only if one or more of its parents are true. This restriction makes it easy to find a value assignment $\boldsymbol{w}$ such that $P(\boldsymbol{w}|\boldsymbol{v}) > 0$. Although this theorem could be demonstrated by adapting the proof for Theorem 5.1, it is useful to show that the best-small plausibility criterion has a correlate in probabilistic reasoning.

The reduction from independent abduction problems using best-small works as follows. Each $h \in H_{all}$ is mapped to a "hypothesis" variable. Each $d \in D_{all}$ is mapped to a "data" variable that is true if and only if one or more of the hypothesis variables corresponding to $e^{-1}(d)$ are true, i.e., an OR interaction. The *a priori* probabilities of the hypothesis variables being true must be between 0 and .5, and are ordered according to the plausibilities in the abduction problem. Initializing all the data variables to *true* sets up the problem. The MPE for this belief revision problem corresponds to a best explanation for the best-small problem. Because finding a best explanation is NP-hard, finding the MPE must be NP-hard even for belief networks that only contain OR interactions.

5.2 Ordered Abduction Problems

Our proofs of Theorem 5.1 depend on the fact that some individual hypotheses have similar plausibilities to other hypotheses. It turns out that finding a best explanation using best-small is tractable if the plausibilities of individual hypotheses are all different from each other and if their plausibilities are totally ordered.

Formally, an abduction problem is *ordered* if:

$$\forall h, h' \in H_{all} \; (h \neq h' \rightarrow (pl(h) < pl(h') \vee pl(h) > pl(h')))$$

Again, let $n = |D_{all}| + |H_{all}|$.

Theorem 5.3 *For the class of ordered monotonic abduction problems using the best-small plausibility criterion, there is an $O(nC_e + nC_{pl} + n^2)$ algorithm for finding a best explanation.*

Algorithm 2 performs this task within this order of complexity. It is same as Algorithm 1 except that the loop considers the individual hypotheses from least to most plausible. The explanation that Algorithm 2 finds is a best explanation because no other explanation can have more plausible individual hypotheses; the algorithm always chooses the least plausible individual hypotheses to remove. Of course, Algorithm 2 also finds a best explanation for ordered independent abduction problems.

As with Algorithm 1, it is possible to use e^{-1} advantageously. The working hypothesis W can be initialized to include the most plausible individual hypotheses from each $e^{-1}(d)$, i.e., because of monotonic interactions, sufficient hypotheses from $e^{-1}(d)$ must be chosen so that d is explained.

Algorithm 2 is an adaptation of the hypothesis assembly algorithm described in Josephson et al. [13], and is a serial version of the parallel parsimony algorithm described in Goel [11]. In Figure 1 assuming the independence constraint, this algorithm would find $\{h_1, h_3, h_4\}$, which is one of the two best explanations.

As in our example, there might be more than one explanation because best-small in general imposes a partial ordering on the plausibilities of composite hypotheses. Suppose that an ordered monotonic abduction problem had only one best explanation according to best-small. Because Algorithm 2 is guaranteed to find a best explanation, then it will find the one best explanation.

Corollary 5.4 *For the class of ordered monotonic abduction problems using the best-small plausibility criterion, if there is exactly one best explanation, then there is an $O(nC_e + nC_{pl} + n^2)$ algorithm for finding the best explanation.*

This can be informally restated as: *In a well-behaved abduction problem, if it is known that some explanation is clearly the best explanation, then it is tractable to find it.* Unfortunately, it is difficult to determine if some explanation is clearly the best explanation.

Theorem 5.5 *For the class of ordered independent abduction problems using the best-small plausibility criterion, given a best explanation, it is NP-complete to determine whether there is another best explanation.*

W stands for the working composite hypothesis.
Nil is returned if no explanation exists.

Determine whether an explanation exists.
If $e(H_{all}) \neq D_{all}$ then
    Return nil

Find a best explanation.
$W \leftarrow H_{all}$
For each $h \in H_{all}$ from least to most plausible
    If $e(W \setminus \{h\}) = D_{all}$ then
        $W \leftarrow W \setminus \{h\}$
Return W

Algorithm 2: Finding a Best Explanation in Ordered Independent and Monotonic Abduction Problems Using the Best-Small Plausibility Criterion

We have proved this by a reduction from the special class of independent incompatibility abduction problems in which each individual hypothesis is in exactly one incompatible pair. Assuming n incompatible pairs, the best-small problem is set up so that one hypothesis out of each pair must be chosen, and so that extra hypotheses plus the most plausible element of each pair is a best explanation of size $n + 2$. In our reduction, any other best-small best explanation in this reduction must be of size $n + 1$ and include an explanation for the incompatibility problem. Thus, even for ordered independent abduction problems, it is intractable to find all the best explanations, or even enumerate some number of them.

As a consequence, it does not become tractable to find the MPE for ordered abduction problems. The proof for the previous theorem can be easily adapted so that any explanation of size $n + 1$ will be more probable than any explanation of size $n + 2$.

From these theorems, we can describe what kinds of mistakes will be made by Algorithm 2. While the explanation this algorithm finds will match up qualitatively to any other explanation, there might be other "qualitatively best" explanations, which might be judged better based on more precise plausibility information.

Table 2 summarizes the results of this and the previous section.

6 A Real-World Application of Abduction—Red Blood Antibody Identification

6.1 Description of the Domain

The RED expert system performs in the domain of blood bank antibody analysis [28]. One of the jobs done by a blood-bank technologist is to identify antibodies in a patient's serum that can react to antigens that might appear on red blood cells. This is typically

Table 2: Computational Complexity of Finding Best Explanations Using the Best-Small Plausibility Criterion

class of problems	condition to achieve	
	finding a best explanation	finding more than one best explanation
ordered independent/monotonic	P	NP
unordered independent/monotonic	NP	NP
incompatibility	NP	NP
cancellation	NP	NP

P = known polynomial algorithm NP = NP-hard

done by combining, under different test conditions, samples of patient serum with samples of red blood cells known to express certain antigens. Some of these combinations might show reactions. The presence of certain antibodies in the patient serum accounts for certain reactions. The reactions are additive in the sense that if the presence of one antibody explains one reaction, and presence of another antibody explains another, then the presence of both antibodies explains both reactions. If each antibody can account for a weak result in some reaction, then the presence of both can account for a stronger result in that reaction. Also, *some pairs of antibodies cannot occur together*. RED's task is to decide which antibodies are present, given a certain reaction pattern. RED takes into account about 30 of the most common antibodies.

6.2 Relationship to Classes of Abduction Problems

We now examine how this task can be categorized within the classes of abduction problems discussed in this paper.

Independent. The additive nature of the reactions means that for separate reactions and compatible hypotheses, the independence constraint is met. However, since independent abduction problems do not allow for parts of data to be explained, they cannot describe additivity of reaction strengths.

Monotonic. If we view a weak result for some reaction as a separate result from a strong result for the same reaction, then we can say that the phenomenon of additive reaction strengths falls into the class of monotonic abduction problems. That is, each of two antibodies alone might explain a weak reaction. Together, they would explain either a weak reaction or a strong reaction.

Incompatibility. In this domain, some antibodies are incompatible with others. Also, for each antibody, RED distinguishes between two different, incompatible ways that it can react. Thus, red blood antibody identification is clearly outside of monotonic abduction problems and within the intractable class of incompatibility abduction problems. Below, we discuss why this is not usually a difficulty in this domain.

Cancellation. No cancellation interactions take place in this domain.

Ordered. RED rates the plausibility of the presence of an antibody on a 7-point qualitative scale. Because there are about 60 antibody subtypes, the same plausibility rating is given to several antibodies. Strictly speaking, this takes the problem out of ordered abduction problems, but we describe below why this is not usually a problem.

Incompatibility relationships and lack of plausibility ordering do not usually create difficulties in this domain for the following reasons. One is that most antibodies are usually ruled out before any composite hypotheses are considered, i.e., the evidence indicates that the antibodies cannot reasonably be part of any composite hypotheses. The more antibodies that are ruled out, the more likely that the remaining antibodies contain no or few incompatibilities, and resemble an ordered abduction problem.

Another reason is that the reaction testing is designed to discriminate between the antibodies. Thus, an antibody that is present usually explains some reaction more plausibly than any other antibody. An antibody that is not present is unlikely to have clear evidence in its favor and is usually superfluous in the context of equally or higher rated antibodies.

A final reason is that it is rare to have more than a few antibodies. Other antibodies that are rated lower than these antibodies are easily eliminated.

In rare cases, though, these reasons do not apply with the result that RED has difficulties with incompatible pairs or unordered hypotheses, or that RED selects an explanation with many antibodies whereas the preferred answer contains a smaller number of individually less plausible antibodies [27].

7 Discussion

We have discovered one restricted class of abduction problems in which it is tractable to find the best explanation. In this class, there can be no incompatibility relationships or cancellation interactions, the plausibilities of the individual hypotheses are all different from each other, and there must be exactly one best explanation according to the best-small plausibility criterion. Unfortunately, it is intractable to determine whether there is more than one best explanation in ordered abduction problems. However, it is still tractable to find one of the best-small best explanations in ordered monotonic abduction problems.

For abduction in general, however, our results are not encouraging. We believe that few domains satisfy the independent or monotonic property, i.e., they usually have incompatibility relationships and cancellation interactions. Requiring the most plausible explanation appears to guarantee intractability for abduction. It is important to note that these difficulties result from the nature of abduction problems, and not the representations or algorithms being used to solve the problem. *These problems are hard no matter what representation or algorithm is used.*

Fortunately, there are several mitigating factors that might hold for specific domains. One factor is that incompatibility relationships and cancellation interactions might be sufficiently sparse so that it is not expensive to search for explanations. However, only $O(n)$ incompatibilities or cancellations are sufficient to lead to intractability, and the maximum plausibility requirement still remains a difficulty.

Another factor, as discussed in Section 1, is that some constraint might guarantee a polynomial search space, e.g., a limit on the size of hypotheses or sufficient knowledge to rule

out most individual hypotheses. For example, if rule-out knowledge can reduce the number of individual hypotheses from h to $\log h$, then the problem is tractable. It is important to note that such factors do not simply call for "more knowledge," but knowledge of the right type, e.g., *rule-out* knowledge. Additional knowledge *per se* does not reduce complexity. For example, more knowledge about incompatibilities or cancellations makes abduction harder.

The abductive reasoning of the RED expert system works because of these factors. The size of the right answer is usually small, and rule-out knowledge is able to eliminate many hypotheses. RED is able to avoid exhaustive search because the non-ruled-out hypotheses are close to an ordered monotonic abduction problem.

If there are no tractable algorithms for a class of abduction problems, then there is no choice but do abduction heuristically (unless one is willing to wait for a very long time). This poses a challenge to researchers who attempt to deal with abductive inference—provide a characterization that respects the classic criteria of good explanations (parsimony, coverage, consistency, and plausibility), but avoids the computational pitfalls that beset solutions attempting to optimize these criteria. We believe this will lead to the adoption of a more naturalistic or satisficing conceptualization of abduction [14, 29], in which the final explanation is not guaranteed to be optimal, e.g., it might not explain some data. Perhaps one mark of intelligence is being able to act despite the lack of optimal solutions.

Our results show that abduction, characterized as finding the most plausible composite hypothesis that explains all the data, is generally an intractable problem. Thus, it is futile to hope for a tractable algorithm that produces optimal answers for all kinds of abduction problems. To be solved efficiently, an abduction problem must have certain features that *make it tractable, and a reasoning method* that takes advantage of those features. Understanding abduction, as for any portion of intelligence, requires a theory of reasoning that takes care for the practicality of computations.

References

[1] D. Allemang, M. C. Tanner, T. Bylander, and J. R. Josephson. On the computational complexity of hypothesis assembly. In *Proc. Tenth Int. Joint Conf. on Artificial Intelligence*, pages 1112–1117, Milan, 1987.

[2] T. Bylander, D. Allemang, M. C. Tanner, and J. R. Josephson. Some results concerning the computational complexity of abduction. In *Proc. First Int. Conf. on Principles of Knowledge Representation and Reasoning*, pages 44–54, Toronto, 1989.

[3] E. Charniak and D. McDermott. *Introduction to Artificial Intelligence*. Addison-Wesley, Reading, MA, 1985.

[4] G. Cooper. The computational complexity of probablistic inference using Bayesian belief networks. *Artificial Intelligence*, 42(2-3):393–405, 1990.

[5] R. Davis and W. Hamscher. Model-based reasoning: Troubleshooting. In H. E. Shrobe, editor, *Exploring Artificial Intelligence*, pages 297–346. Morgan Kaufmann, San Mateo, CA, 1988.

[6] J. de Kleer and B. C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, 1987.

[7] J. de Kleer and B. C. Williams. Diagnosis with behavioral modes. In *Proc. Eleventh Joint Int. Conf. on Artificial Intelligence*, pages 1324–1330, Detroit, 1989.

[8] D. Dvorak and B. Kuipers. Model-based monitoring of dynamic systems. In *Proc. Eleventh Joint Int. Conf. on Artificial Intelligence*, pages 1238–1243, Detroit, 1989.

[9] L. Eshelman. MOLE: A knowledge-acquisition tool for cover-and-differentiate systems. In S. Marcus, editor, *Automating Knowledge Acquisition for Expert Systems*, pages 37–80. Kluwer, Boston, 1988.

[10] M. R. Garey and D. S. Johnson. *Computers and Intractability*. W. H. Freeman, New York, 1979.

[11] A. Goel, P. Sadayappan, and J. R. Josephson. Concurrent synthesis of composite explanatory hypotheses. In *Proc. Seventeeth Int. Conf. on Parallel Processing*, pages 156–160, St. Charles, IL, 1988.

[12] J. R. Hobbs, M. Stickel, P. Martin, and D. Edwards. Interpretation as abduction. In *Proc. 26th Annual Meeting of the Assoc. for Computational Linguistics*, pages 95–103, Buffalo, NY, 1988.

[13] J. R. Josephson, B. Chandrasekaran, J. W. Smith, and M. C. Tanner. A mechanism for forming composite explanatory hypotheses. *IEEE Trans. Systems, Man, and Cybernetics*, 17(3):445–454, 1987.

[14] J. R. Josephson and A. Goel. Practical abduction. Technical report, Lab. for AI Research, CIS Dept., Ohio State Univ., Columbus, OH, 1990.

[15] H. J. Levesque. A knowledge-level account of abduction. In *Proc. Eleventh Joint Int. Conf. on Artificial Intelligence*, pages 1061–1067, Detroit, 1989.

[16] R. A. Miller, J. E. Pople, and J. D. Myers. INTERNIST-I, An experimental computer-based diagnostic consultant for general internal medicine. *New England J. of Medicine*, 307:468–476, 1982.

[17] R. S. Patil, P. Szolovits, and W. B. Schwartz. Modeling knowledge of the patient in acid-base and electrolyte disorders. In P. Szolovits, editor, *Artificial Intelligence in Medicine*, pages 191–226. Westview Press, Boulder, CO, 1982.

[18] J. Pearl. Fusion, propagation, and structuring in belief networks. *Artificial Intelligence*, 29(3):241–288, 1986.

[19] J. Pearl. Distributed revision of composite beliefs. *Artificial Intelligence*, 33(2):173–215, 1987.

[20] C. S. Peirce. Abduction and induction. In J. Buchler, editor, *Philosophical Writings of Peirce*, chapter 11, pages 150–156. Dover, New York, 1955.

[21] Y. Peng and J. A. Reggia. A probabilistic causal model for diagnostic problem solving. *IEEE Trans. on Systems, Man, and Cybernetics*, 17(2):146–162, 1987. Part II appears in SMC-17(3):395–406.

[22] D. Poole. Explanation and prediction: An architecture for default and abductive reasoning. *Computational Intelligence*, 5(2):97–110, 1989.

[23] H. E. Pople. On the mechanization of abductive logic. In *Proc. Third Int. Joint Conf. on Artificial Intelligence*, pages 147–152, Stanford, 1973.

[24] H. E. Pople. The formation of composite hypotheses in diagnostic problem solving: An exercise in synthetic reasoning. In *Proc. Fifth Int. Joint Conf. on Artificial Intelligence*, pages 1030–1037, Cambridge, MA, 1977.

[25] J. A. Reggia, D. S. Nau, and P. Wang. Diagnostic expert systems based on a set covering

model. *Int. J. Man-Machine Studies*, 19(5):437–460, 1983.

[26] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[27] J. W. Smith, J. R. Josephson, M. C. Tanner, J. R. Svirbely, and P. Strohm. Problem solving in red cell antibody identification: RED's performance on 20 cases. Technical report, LAIR, CIS Dept., Ohio State Univ., Columbus, OH, 1986.

[28] J. W. Smith, J. R. Svirbely, C. A. Evans, P. Strohm, J. R. Josephson, and M. C. Tanner. RED: A red-cell antibody identification expert module. *J. of Medical Systems*, 9(3):121–138, 1985.

[29] J. Sticklen. Distributed abduction. In R. Huber, C. Kulikowski, J. M. Krivine, and J. M. David, editors, *Artificial Intelligence in Scientific Computation: Towards Second Generation Systems*. J. C. Baltzer, Basel, Switzerland, 1989.

[30] P. Struss and O. Dressler. "Physical negation"—Integrating fault models into the general diagnostic engine. In *Proc. Eleventh Joint Int. Conf. on Artificial Intelligence*, pages 1318–1323, Detroit, 1989.

[31] P. Thagard. *Computational Philosophy of Science*. MIT Press, Cambridge, MA, 1988.

[32] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. of Computing*, 8(3):410–421, 1979.

A Proofs of Theorems

In this appendix, we provide a proof for each theorem in the paper. We assume that the functions e and pl are tractable with time complexities $O(C_e)$, and $O(C_{pl})$ in the size of an abduction problem, respectively (see Section 3). The reader is forewarned that many of the reduction proofs do not provide direct insight on the intuitive reasons underlying the complexity results. The proof of Theorem 4.3 is given after Theorem 4.5.

Theorem 4.1 *For the class of independent abduction problems, it is #P-complete to determine the number of explanations.*

This is in #P because each composite hypothesis $H \subseteq H_{all}$ can be generated in nondeterministic polynomial time, and it is easy to check if H is complete and parsimonious.

It is possible to reduce from determining the number of minimal vertex covers [32] to determining the number of explanations. Given a graph G with vertices V and edges E, a minimal vertex cover is a minimal subset of vertices $V' \subseteq V$ such that every edge is connected to a vertex in V'. An independent abduction problem can be constructed from G as follows:

$$\begin{aligned} D_{all} &= E \\ H_{all} &= V \\ e(H) &= \{(u,v) \in E \mid u \in H \vee v \in H\} \\ pl(H) &= \text{anything} \end{aligned}$$

In this construction, H is an explanation iff it corresponds to a minimal vertex cover.

Theorem 4.2 *For the class of independent abduction problems, there is an $O(nC_e + n^2)$ algorithm for finding an explanation, if one exists.*

Theorem 4.4 *For the class of monotonic abduction problems, there is an $O(nC_e + n^2)$ algorithm for finding an explanation, if one exists.*

For these theorems, $n = |D_{all}| + |H_{all}|$. Because the monotonic constraint is more general than the independent constraint, our discussion is oriented for Theorem 4.4. First we consider the correctness of Algorithm 1, and then consider its complexity.

The first conditional determines whether an explanation exists. Because of the monotonic constraint, H_{all} must explain as much or more than any other composite hypothesis. Hence, if H_{all} is not a complete composite hypothesis, then no composite hypothesis is complete. If H_{all} passes this test, then initializing W to H_{all} ensures that W starts off as a complete composite hypothesis.

Within the loop, W remains a complete composite hypothesis because no element is removed if it leads to explaining less than D_{all}. The fact that the result W is also parsimonious can be shown by contradiction. Suppose that $H \subset W$ and $e(H) = D_{all}$. Then, because the problem is monotonic, $H \subseteq H' \rightarrow e(H) = e(H') = e(W)$. In particular, for each $h \in W \setminus H$, $e(H) = e(W \setminus \{h\}) = e(W)$. However, the loop would have removed h from W (or any superset of W) in just this case, implying that $h \notin W$, which is a contradiction. Thus, the loop produces a complete and parsimonious composite hypothesis, i.e., an explanation.

Now consider the complexity of Algorithm 1. We assume sets are represented as bit vectors. Let $k = |D_{all}|$ and $l = |H_{all}|$.

The conditional makes one call to e ($O(C_e)$) and checks whether its answer is equal to D_{all} ($O(k)$ steps). Assigning H_{all} to W takes $O(l)$ steps.

Since $|H_{all}| = l$, the loop iterates l times. The loop performs l evaluations of e ($O(C_e)$ each), l set comparisons ($O(k)$ each), and up to l set differences of single elements ($O(1)$ each). Thus the complexity of algorithm is $O(lC_e + kl + k + 2l)$. Since both k and l are less than n, $O(nC_e + n^2)$ clearly bounds the complexity of Algorithm 1.

Theorem 4.5 *For the class of independent incompatibility abduction problems, it is NP-complete to determine whether there is an explanation.*

We prove the NP-completeness of this problem by reducing from 3SAT [10]: given a statement in propositional calculus in conjunctive normal form, in which each term has at most three factors, find an assignment of variables that makes the statement true.

Let S be a statement in propositional calculus in 3SAT form. Let $U = \{u_1, u_2, \ldots, u_m\}$ be the variables used in S. Let n be the number of terms in S. An equivalent independent incompatibility abduction problem $\mathcal{P} = \langle D_{all}, H_{all}, e, pl, \mathcal{I} \rangle$ can be constructed by:

$$
\begin{aligned}
D_{all} &= \{d_1, d_2, \ldots, d_n\} \\
H_{all} &= \{h_1, h'_1, h_2, h'_2, \ldots, h_m, h'_m\} \\
e(h_i) &= \{d_j \mid u_i \text{ is a factor in the } j\text{th term}\} \\
e(h'_i) &= \{d_j \mid \neg u_i \text{ is a factor in the } j\text{th term}\} \\
pl(H) &= \text{anything} \\
\mathcal{I} &= \{\{h_1, h'_1\}, \{h_2, h'_2\}, \ldots, \{h_m, h'_m\}\}
\end{aligned}
$$

If H is a complete composite hypothesis for $\mathcal{P}$, then S is satisfied by the following assignment.

$$
u_i = \begin{cases} \text{true} & \text{if } h_i \in H \\ \text{false} & \text{otherwise} \end{cases}
$$

Consider the jth term in S. Since $d_j \in e(H)$, there is some i such that either h_i or h'_i is in H and explains d_j. If $h_i \in H$ and $d_j \in e(h_i)$, then u_i is a factor in the jth term of S and by the assignment above, the term is satisfied. If $h'_i \in H$ and $d_j \in e(h'_i)$, then $h_i \notin H$ and $\neg u_i$ is a factor in the jth term of S, so by the assignment above, the term is satisfied. Since j was arbitrary, it must be the case that all terms in S will be satisfied by the above assignment.

If S is satisfied by a value assignment $U' \subseteq U$ indicating:

$$u_i = \begin{cases} \text{true} & \text{if } u \in U' \\ \text{false} & \text{otherwise} \end{cases}$$

then $H = \{h_i \mid u_i \in U'\} \cup \{h'_i \mid u_i \notin U'\}$ will be a complete composite hypothesis for $\mathcal{P}$.

An explanation for $\mathcal{P}$ exists if and only if a complete composite hypothesis for $\mathcal{P}$ exists. Thus, 3SAT problems reduce to independent incompatibility abduction problems. Incompatibility abduction problems are clearly in NP since it is easy to guess a composite hypothesis H and to test whether $e(H) = D_{all}$. Thus, it is NP-complete to determine whether an independent incompatibility abduction problem has an explanation.

Below, we reduce this class of problems to a number of other classes. For convenience, we shall assume that these abduction problems have the same form as the $\mathcal{P}$ constructed above: the problem is independent except for incompatibilities, each $h \in H_{all}$ is an element of exactly one $I \in \mathcal{I}$, and $|e^{-1}(d)| \leq 3$ for each $d \in D_{all}$. We refer to this special class of independent incompatibility abduction problems as *3SAT abduction problems.*

Theorem 4.3 *For the class of monotonic abduction problems, given a set of explanations, it is NP-complete to determine whether an additional explanation exists.*

This can be reduced from 3SAT abduction problems. Let $\mathcal{P} = \langle D_{all}, H_{all}, e, pl, \mathcal{I} \rangle$ be a 3SAT abduction problem, and let $\mathcal{P}' = \langle D'_{all}, H'_{all}, e', pl' \rangle$ be a monotonic abduction problem constructed from P as follows.

$$\begin{aligned} D'_{all} &= D_{all} \\ H'_{all} &= H_{all} \\ e'(H) &= \begin{cases} D_{all} & \text{if } \exists I \in \mathcal{I}\,(I \subseteq H) \\ e(H) & \text{otherwise} \end{cases} \\ pl'(H) &= pl(H) \end{aligned}$$

It turns out that $\mathcal{I}$ is a set of explanations for $\mathcal{P}'$. Consequently, any other explanation can only have at most one hypothesis from each pair $I \in \mathcal{I}$. Thus, it would also be a explanation for $\mathcal{P}$.

Theorem 4.7 *For the class of diagnosis problems, relative to the complexity of determining whether a composite hypothesis is consistent with* SDUOBS, *it is NP-complete to determine whether a diagnosis exists.*

To determine whether a solution exists for a diagnosis problem, it is sufficient to exhibit a subset of components $\Delta \subseteq$ COMPONENTS such that

$$\text{SD} \cup \text{OBS} \cup \{\text{AB}(c) \mid c \in \Delta\} \cup \{\neg\text{AB}(c) \mid c \in \text{COMPONENTS} \setminus \Delta\}$$

is consistent. Hence, diagnosis problems are in NP relative to the complexity of determining whether a composite hypothesis is consistent with SD∪OBS.

Determining whether a diagnosis exists can be reduced from finding explanations for 3SAT abduction problems. Let $\mathcal{P} = \langle D_{all}, H_{all}, e, pl, \mathcal{I} \rangle$ be a 3SAT abduction problem, and let $\mathcal{P}' = \langle \text{SD}, \text{OBS}, \text{COMPONENTS} \rangle$ be a diagnosis problem constructed from $\mathcal{P}$ as follows.

$$\begin{aligned} \text{SD} &= \{\text{MANIFEST}(d) \rightarrow \textstyle\bigvee_{h \in e^{-1}(d)} \text{AB}(h) \mid d \in D_{all}\} \cup \\ &\quad \{\text{AB}(h) \rightarrow \neg\text{AB}(h') \mid \{h, h'\} \in \mathcal{I}\} \\ \text{OBS} &= \{\text{MANIFEST}(d) \mid d \in D_{all}\} \\ \text{COMPONENTS} &= H_{all} \end{aligned}$$

An explanation for $\mathcal{P}$ exists iff there is a complete composite hypothesis H in $\mathcal{P}$, which is equivalent to whether:

$$\text{SD} \cup \text{OBS} \cup \{\text{AB}(h) \mid h \in H\} \cup \{\neg\text{AB}(h) \mid h \in H_{all} \setminus H\}$$

is consistent for $\mathcal{P}'$. Clearly, some $\text{AB}(h)$ must be true for each observation $\text{MANIFEST}(d)$. Also, $\text{AB}(h)$ and $\text{AB}(h')$ cannot be true at the same time if $\{h, h'\} \in \mathcal{I}$.

Theorem 4.8 *For the class of belief revision problems, it is NP-complete to determine whether there is a value assignment $\boldsymbol{w}$ to the variables $\boldsymbol{W}$ such that $P(\boldsymbol{w}|\boldsymbol{v}) > 0$.*

Cooper [4] showed that it is NP-complete to determine whether $P(X = true) > 0$ for a given variable X. To prove the above theorem, simply let $\boldsymbol{V} = \{X\}$ and $\boldsymbol{v}$ consist of $X = true$. Determining whether there is a value assignment $\boldsymbol{w}$ such that $P(\boldsymbol{w}|\boldsymbol{v}) > 0$ is equivalent to determining whether $P(X = true) > 0$.

Theorem 4.9 *For the class of cancellation abduction problems, it is NP-complete to determine whether an explanation exists.*

This can be shown by reduction from 3SAT abduction problems. Let $\mathcal{P} = \langle D_{all}, H_{all}, e, pl, \mathcal{I} \rangle$ be a 3SAT abduction problem. An equivalent cancellation abduction problem $\mathcal{P}' = \langle D'_{all}, H'_{all}, e', pl', e'_+, e'_- \rangle$ can be constructed by:

$$\begin{aligned} D'_{all} &= D_{all} \cup \{d_I \mid I \in \mathcal{I}\}^8 \\ H'_{all} &= H_{all} \cup \{h', h''\} \\ e'_+(h) &= \begin{cases} e(h) & \text{if } h \in H_{all} \\ \{d_I \mid I \in \mathcal{I}\} & \text{if } h \in \{h', h''\} \end{cases} \\ e'_-(h) &= \begin{cases} \{d_I \mid h \in I\} & \text{if } h \in H_{all} \\ \emptyset & \text{if } h \in \{h', h''\} \end{cases} \\ pl'(H) &= pl(H) \end{aligned}$$

Cancellation interactions are created so that each incompatible pair in $\mathcal{P}$ effectively become incompatible in $\mathcal{P}'$. For all $I \in \mathcal{I}$ and $H' \subseteq H'_{all}$, if $I \subseteq H'$, then $d_I \notin e'(H')$. Thus, no such H' can be an explanation. Hence, $\mathcal{P}'$ has a complete composite hypothesis iff $\mathcal{P}$ has a complete composite hypothesis. In particular, $H \subseteq H_{all}$ is a complete composite hypothesis for $\mathcal{P}$ if and only if $H \cup \{h', h''\}$ is a complete composite hypothesis for $\mathcal{P}'$. Consequently, $\mathcal{P}$ has an explanation if and only if $\mathcal{P}'$ has an explanation.

[8] This means that D'_{all} has a datum corresponding to each $I \in \mathcal{I}$, notated as d_I.

Theorem 4.11 *For the class of cancellation abduction problems, it is coNP-complete to determine whether a complete composite hypothesis is parsimonious.*

That is, it is NP-complete to determine whether a complete composite hypothesis is not parsimonious. This can be shown by reduction from 3SAT abduction problems. Let $\mathcal{P} = \langle D_{all}, H_{all}, e, pl, \mathcal{I} \rangle$ be a 3SAT abduction problem, and let $\mathcal{P}' = \langle D'_{all}, H'_{all}, e', pl', e'_+, e'_- \rangle$ be a cancellation abduction problem constructed from $\mathcal{P}$ as follows:

$$
\begin{aligned}
H'_{all} &= H_{all} \cup \{h', h'', h^*, h^{**}\} \\
D'_{all} &= D_{all} \cup \{d_I \mid I \in \mathcal{I}\} \cup \{d_h \mid h \in H_{all}\} \\
e'_+(h) &= \begin{cases} e(h) \cup d_h & \text{if } h \in H_{all} \\ \{d_I \mid I \in \mathcal{I}\} & \text{if } h \in \{h', h''\} \\ D_{all} \cup \{d_I \mid I \in \mathcal{I}\} & \text{if } h = h^* \\ \{d_h \mid h \in H_{all}\} & \text{if } h = h^{**} \end{cases} \\
e'_-(h) &= \begin{cases} \{d_I \mid h \in I\} & \text{if } h \in H_{all} \\ \{d_h \mid h \in H_{all}\} & \text{if } h = h^* \\ \emptyset & \text{if } h \in \{h', h'', h^{**}\} \end{cases} \\
pl'(H) &= pl(H)
\end{aligned}
$$

This construction is similar to the previous one except that additional data and hypotheses are included so that H'_{all} is a complete composite hypothesis. However, any other complete composite hypothesis (which obviously must be a proper subset of H'_{all}) cannot include h^* and must satisfy cancellation interactions equivalent to $\mathcal{P}$'s incompatibilities. In particular, $H \subseteq H_{all}$ is a complete composite hypothesis for $\mathcal{P}$ if and only if $H \cup \{h', h'', h^{**}\}$ is a complete composite hypothesis for $\mathcal{P}'$.

Theorem 5.1 *For the class of independent abduction problems using the best-small plausibility criterion, it is NP-hard to find a best explanation.*

This can be shown by reduction from 3SAT abduction problems. Let $\mathcal{P} = \langle D_{all}, H_{all}, e, pl, \mathcal{I} \rangle$ be a 3SAT abduction problem, and let $\mathcal{P}' = \langle D'_{all}, H'_{all}, e', pl' \rangle$ be an independent abduction problem using best-small constructed from $\mathcal{P}$ as follows:

Let f_1 be a function from $\mathcal{I}$ to H_{all}, such that $\forall I \in \mathcal{I}\ (f_1(I) \in I)$, i.e., f_1 chooses one hypothesis from each pair in $\mathcal{I}$. Let H_1 be the set of hypotheses that f_1 chooses, i.e, $H_1 = \bigcup_{I \in \mathcal{I}} \{f_1(I)\}$. Let f_2 be another function from $\mathcal{I}$ to H_{all}, such that f_2 chooses the other hypothesis from each pair in I. Now define $\mathcal{P}'$ as:

$$
\begin{aligned}
D'_{all} &= D_{all} \cup \{d_I \mid I \in \mathcal{I}\} \cup \{d'\} \\
H'_{all} &= H_{all} \cup \{h', h''\} \\
e'(h) &= \begin{cases} e(h) \cup \{d_I \mid h \in I\} & \text{if } h \in H_{all} \\ \{d'\} \cup D_{all} \backslash e(H_1) & \text{if } h = h' \\ \{d'\} & \text{if } h = h'' \end{cases} \\
&\forall I \in \mathcal{I}\ (pl'(f_1(I)) = pl'(f_2(I))) \\
&\forall h \in H_{all}\ (pl'(h) < pl'(h') < pl'(h''))
\end{aligned}
$$

The remaining orderings of pl' do not matter. Let $n = |\mathcal{I}|$. Note that one hypothesis out of each $I \in \mathcal{I}$ must be chosen to explain all the d_I. Either h' or h'' must be chosen

to explain d'. Hence, explanations must be of size $n+1$ or larger. Now $H = H_1 \cup \{h'\}$ is an explanation of size $n+1$, where $n = |\mathcal{I}|$. Because h' makes h'' superfluous, any other explanation of greater size must include more than n elements of H_{all}. According to best-small, this would match h' of H against a lower ranking hypothesis, so H must be better than any larger explanation. No other explanation can be smaller, so to get a better explanation according to best-small, h'' must be chosen, and only one hypothesis out of each $I \in \mathcal{I}$ can be accepted so that they explain D_{all}, i.e., a solution to the 3SAT abduction problem $\mathcal{P}$. Such an explanation would be a best explanation and also show that H is not a best explanation. Hence, finding the best explanation would solve the 3SAT abduction problem. Thus, finding a best explanation for independent abduction problems using the best-small plausibility criterion is NP-hard.

Theorem 5.2 *For the class of belief revision problems restricted to OR interactions, it is NP-hard to find the most probable explanation.*

We reduce from finding a best explanation in independent abduction problems using the best-small plausibility criterion. Let $\mathcal{P} = \langle D_{all}, H_{all}, e, \boldsymbol{pl} \rangle$ be an independent abduction problem where $\boldsymbol{pl}$ satisfies the best-small plausibility criterion. A belief revision problem $\mathcal{P}'$ that preserves the orderings among complete composite hypotheses determined by $\boldsymbol{pl}$, but might create additional orderings, is:

$$
\begin{aligned}
&\boldsymbol{W} = \{X_d \mid d \in D_{all}\} \cup \{X_h \mid h \in H_{all}\} \\
&\boldsymbol{V} = \{X_d \mid d \in D_{all}\} \\
&S_d = \{X_h \mid d \in e(h)\} \\
&S_h = \emptyset \\
&\boldsymbol{v} = \{X = \mathit{true} \mid X \in \boldsymbol{V}\} \\
&\forall \boldsymbol{s}_d \, ((\boldsymbol{s}_d \rightarrow \exists X_h \, (d \in e(h) \wedge X_h = \mathit{true})) \rightarrow P(X_d = \mathit{true} | \boldsymbol{s}_d) = 1) \\
&\forall \boldsymbol{s}_d \, ((\boldsymbol{s}_d \rightarrow \forall X_h \, (d \in e(h) \rightarrow X_h = \mathit{false})) \rightarrow P(X_d = \mathit{true} | \boldsymbol{s}_d) = 0) \\
&\forall X_h \, (0 < P(X_h = \mathit{true}) < .5) \\
&\forall X_h, X_{h'} \, (\boldsymbol{pl}(h) < \boldsymbol{pl}(h') \rightarrow P(X_h = \mathit{true}) < P(X_{h'} = \mathit{true})) \\
&\forall X_h, X_{h'} \, (\boldsymbol{pl}(h) = \boldsymbol{pl}(h') \rightarrow P(X_h = \mathit{true}) = P(X_{h'} = \mathit{true}))
\end{aligned}
$$

Note that $P(X_d = \mathit{true}) = 1$ iff there is some h such that $d \in e(h)$ and $P(X_h = \mathit{true}) = 1$. The network thus consists solely of OR interactions. As a consequence, the conditional probabilities can be concisely represented. This ensures that the size of $\mathcal{P}'$ is polynomial in the size of $\mathcal{P}$.

For convenience, we will use $P(H|D)$ to denote the probability that X_h is *true* for $h \in H$ and X_h is *false* for $h \in H_{all} \backslash H$, given that X_d is *true* for $d \in D$ and X_h is *false* for $d \in D_{all} \backslash D$. If a value assignment $\boldsymbol{w}$ to all the variables $\boldsymbol{W}$ assigns *true* to all X_d and to only X_h such that $h \in H$, it is easy to verify that $P(\boldsymbol{w}|\boldsymbol{v}) = P(H|D_{all})$.

If H is a complete composite hypothesis for the abduction problem $\mathcal{P}$, then H can be compared to other complete composite hypotheses in the belief revision problem $\mathcal{P}'$ using the expression:

$$\prod_{h \in H} P(h) \prod_{h \in H_{all} \backslash H} (1 - P(h))$$

where $P(h)$ denotes $P(X_h = true)$. By Bayes' theorem:

$$P(H|D_{all}) = \frac{P(H)P(D_{all}|H)}{P(D_{all})}$$

Because of the way the conditional probabilities are set up, H implies D_{all}, so $P(D_{all}|H) = 1$. Because $P(D_{all})$ will always be in the denominator, it is sufficient to compare $P(H)$ with $P(H')$ if H' is another complete composite hypotheses. $P(H)$ is calculated by the expression given above. Obviously $P(H|D_{all}) > 0$ if H is a complete composite hypothesis.

If H is not a complete composite hypothesis, then $P(D_{all}|H) = 0$, and so $P(H|D_{all}) = 0$.[9]

Suppose $H*$ is the composite hypothesis that corresponds to the MPE $w*$ for the belief revision problem. To show that $H*$ is a best explanation, we need to show that $H*$ is complete, $H*$ is parsimonious, and that no other explanation H is better than $H*$ based on the best-small plausibility criterion. From the above discussion, it should be clear that $H*$ is complete.

Now if $H*$ were not parsimonious, then some $h \in H*$ is superfluous, i.e., $H*\backslash\{h\}$ is complete. However, because $1 - P(h) > P(h)$, it would follow that $P(H*\backslash\{h\}) > P(H*)$, which contradicts the fact that $H*$ corresponds to the MPE. Thus, it must be the case that $H*$ is parsimonious. Since $H*$ is also complete, $H*$ is an explanation.

Finally, suppose that another explanation H is better than $H*$ according to the best-small plausibility criterion. Then there exists a 1-1 function m from H to $H*$ that satisfies the following conditions: for each $h \in H$, $P(h) \geq P(m(h))$; and either H is smaller than $H*$, or there exists an $h \in H$ such that $P(h) > P(m(h))$.

Using the function m, a 1-1 function m' from H_{all} to H_{all} can be constructed as follows:

$$m'(h) = \begin{cases} m(h) & \text{if } h \in H \\ m^{-n}(h) & \text{if } h \in H*\backslash H \wedge m^{-n}(h) \in H \wedge m^{-(n+1)}(h) \text{ does not exist} \\ h & \text{otherwise} \end{cases}$$

The domain of m is mapped into the range of m. Whatever is in m's range, but not in m's domain, is inversely mapped into elements in m's domain, but not in m's range. Everything left over is neither in m's range nor domain, and is mapped to itself.

Because of the best-small constraint, the following can be shown for m':

$$\begin{array}{ll} P(h) \geq P(m'(h)) & \text{if } h \in H \\ 1 - P(h) > P(m'(h)) & \text{if } h \in H*\backslash H \wedge \exists h' \in H\ (m(h') = h) \\ 1 - P(h) = 1 - P(m'(h)) & \text{otherwise} \end{array}$$

Since this mapping matches the factors of $P(H)$ to those of $P(H*)$, it shows that $P(H) \geq P(H*)$. Furthermore, best-small guarantees that either $P(h) > P(m(h))$ for some $h \in H$ or that H is smaller, implying there is some $h \notin H$ such that $1 - P(h) > P(m'(h))$. Thus, $P(H) > P(H*)$.

However, this contradicts the fact that $H*$ is the MPE. Thus, it must be the case that the MPE for the belief revision problem $\mathcal{P}'$ corresponds to a best explanation for the abduction problem $\mathcal{P}$. Because finding a best explanation is NP-hard, it is also the case that that finding the MPE is NP-hard, even if the belief network is restricted to OR interactions.

[9] Or undefined if no complete composite hypothesis exists, in which case, no explanation or MPE exists

Theorem 5.3 *For the class of ordered monotonic abduction problems using the best-small plausibility criterion, there is an $O(nC_e + nC_{pl} + n^2)$ algorithm for finding a best explanation.*

where $n = |D_{all}| + |H_{all}|$.

Algorithm 2 is different from Algorithm 1 in only one way: instead of iterating over elements of H_{all} in an arbitrary order in the loop, a specific order is imposed. Thus, by similar arguments as for Theorem 4.4, Algorithm 2 will also find an explanation, if one exists. The change to the loop will result in the addition of at most $O(|H_{all}|)$ evaluations of pl and $O(|H_{all}| \log |H_{all}|)$ steps to sort H_{all} based on pl. Clearly then, the complexity of Algorithm 2 conforms to the order of complexity in the theorem. We now show that the explanation it finds will be a best explanation.

Let W be the explanation it returns. Suppose that, according to the best-small plausibility criterion, there is a better explanation H. Then $|H| \leq |W|$ and there must be a match m of H's elements to W's elements, such that H's elements are just as or more plausible than W's. The matching hypotheses from H to W cannot all have the same plausibility—that would imply that $H \subset W$ because the abduction problem is ordered. However, because W is parsimonious, no proper subset of W can be complete. Thus, at least one element of H must be more plausible than its match in W.

Suppose that the least plausible element $h_1 \in H$ is more plausible than the least plausible element $w_1 \in W$. This implies that all elements of H are more plausible than w_1. Because H is an explanation, that would imply that w_1 is superfluous in the context of higher-rated hypotheses, and that the loop in Algorithm 2 would have removed w_1 from the working hypothesis, implying that $w_1 \notin W$, a contradiction. It must be the case then, that $w_1 \in H$, otherwise H would not match up with W.

Suppose that the m least plausible elements of H are exactly the same as those of W. Call this set W_m. Further suppose that H and W differ on their $m+1$st least plausible elements. Let h_{m+1} and w_{m+1} be these elements, respectively. Consider the hypotheses $W' \subset H_{all}$ that are more plausible than w_{m+1}. Now by the time that w_{m+1} is considered for removal in the loop, the working hypothesis must be $W_m \cup \{w_{m+1}\} \cup W'$. Since w_{m+1} was not found to be superfluous, it must be the case that $W_m \cup W'$ is not complete. Thus, the $m+1$st least plausible element of H cannot be an element of W'. Because $h_{m+1} \neq w_{m+1}$, h_{m+1} must be less plausible than w_{m+1}, which contradicts the supposition that H is a better explanation than W.

Hence, by mathematical induction, no H can be a better explanation than W according to the best-small plausibility criterion. Therefore, for ordered monotonic abduction problems, Algorithm 2 tractably finds a best explanation.

Theorem 5.5 *For the class of ordered independent abduction problems using the best-small plausibility criterion, given a best explanation, it is NP-complete to determine whether an additional best explanation exists.*

This reduction is similar to that of Theorem 5.1. Let $\mathcal{P} = \langle D_{all}, H_{all}, e, pl, \mathcal{I} \rangle$ be a 3SAT abduction problem, and let $\mathcal{P}' = \langle D'_{all}, H'_{all}, e', pl' \rangle$ be an ordered independent abduction problem constructed from $\mathcal{P}$ as follows:

Let f_1 be a function from $\mathcal{I}$ to H_{all}, such that f_1 chooses one hypothesis from each pair in $\mathcal{I}$. Let H_1 be the set of hypotheses that f_1 chooses. Let f_2 be a function from $\mathcal{I}$ to H_{all} that chooses the other hypothesis from each pair in $\mathcal{I}$. Now define $\mathcal{P}'$ as:

$$
\begin{aligned}
D'_{all} &= D_{all} \cup \{d_I \mid I \in \mathcal{I}\} \cup \{d', d''\} \\
H'_{all} &= H_{all} \cup \{h', h'', h^*\} \\
e'(h) &= \begin{cases} e(h) \cup \{d_I \mid h \in I\} & \text{if } h \in H_{all} \\ \{d'\} \cup D_{all} \backslash e(H_1) & \text{if } h = h' \\ \{d''\} & \text{if } h = h'' \\ \{d', d''\} & \text{if } h = h^* \end{cases} \\
&\forall I \in \mathcal{I}\ (pl'(f_1(I)) > pl'(f_2(I))) \\
&\forall h \in H_{all}\ (pl'(h) < pl'(h^*) < pl'(h'') < pl'(h'))
\end{aligned}
$$

The remaining orderings for pl' do not matter. Now $H = H_1 \cup \{h', h''\}$ is a best explanation of size $n + 2$, where $n = |\mathcal{I}|$. Because one hypothesis out of each pair in $\mathcal{I}$ (a total of n hypotheses) is needed to explain $\{d_I \mid I \in \mathcal{I}\}$, and because H includes the more plausible hypothesis of each pair and the two most plausible hypotheses overall, no other explanation of size $n + 2$ or greater can be as good as H. Hence, to construct another best explanation, h' and h'' must be excluded, h^* must be included, and only one hypothesis out of each pair in $\mathcal{I}$ can be accepted, i.e., a solution to the 3SAT abduction problem $\mathcal{P}$. Thus, another best explanation for $\mathcal{P}'$ exists only if $\mathcal{P}$ has an explanation.

Complexity Results for Planning

Tom Bylander
Laboratory for Artificial Intelligence Research
Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210
USA

email: byland@cis.ohio-state.edu
phone: (614)292-5835

December 6, 1990

Submitted to the Twelfth International Joint Conference on Artificial Intelligence
Topic: Automated Reasoning
Subtopic: Planning

Abstract

We describe several computational complexity results for planning, four of which identify tractable planning problems. Our model of planning, called "propositional planning," is simple—conditions within operators are literals or their negations. We consider different planning problems, which are defined by different restrictions on the preconditions and postconditions of operators. Our main results are: Propositional planning is PSPACE-complete, even if operators are restricted to two positive (non-negated) preconditions and two postconditions. It is NP-complete if operators are restricted to positive postconditions, even if operators are restricted to one precondition and one positive postcondition. It is tractable in four restricted cases. One case in which propositional planning is tractable is if each operator is restricted to positive preconditions and one postcondition. The blocks-world problem, slightly modified, is a subclass of this restricted planning problem.

*This research has been supported in part by DARPA/AFOSR contract F49620-89-C-0110 and AFOSR grant 89-0250.

1 Introduction

If the relationship between intelligence and computation is taken seriously, then intelligence cannot be explained by intractable theories because no intelligent creature has the time to perform intractable computations. Nor can intractable theories provide any guarantees about the performance of engineered systems. Presumably, robots don't have the time to perform intractable computations either.

Of course, heuristic theories are a valid approach if partial or approximate solutions are acceptable. However, our purpose is not to consider the relative merits of heuristic theories and tractable theories. Instead, we shall focus on formulating tractable planning problems.

Planning is the reasoning task of finding a sequence of operators that achieve a goal from a given initial state. It is well-known that planning is intractable in general, and that several obstacles stand in the way (Chapman, 1987). However, there are few results that provide clear dividing lines between tractable and intractable planning. Below, we clarify a few of these dividing lines by analyzing the computational complexity of a planning problem and a variety of restricted versions, four of which turn out to be tractable.

Our model of planning, which we call "propositional planning," is impoverished compared to working planners. Preconditions and postconditions of operators are limited to being literals or their negations. An initial state then can be represented as a finite set of literals, indicating that the corresponding conditions are initially true, and that all other relevant conditions are initially false. A goal is represented by two sets of conditions, i.e., the goal is to make the first set of conditions true and the other set false. For convenience, we call these positive and negative goals, respectively. Operators in this model do not have any variables or indirect side effects.

The kinds of restrictions we consider are constraints on the number and kind of pre- and postconditions. Figure 1 illustrates our results, showing which planning problems are PSPACE-complete, NP-hard (at least NP-complete and at most PSPACE-complete), NP-

complete, and polynomial.[1] These results can be summarized as follows:

Propositional planning is PSPACE-complete even if each operator is limited to two positive (non-negated) preconditions and two postconditions.

It is NP-hard if each operator is restricted to one positive precondition and two postconditions.

It is NP-complete if operators are restricted to positive postconditions, even if operators are restricted to one precondition and one positive postcondition.

It is polynomial if each operator is restricted to positive preconditions and one postcondition. The blocks-world problem, slightly modified, is a subclass of this restricted planning problem.

It is polynomial if each operator is restricted to positive preconditions and positive postconditions.

It is polynomial if each operator has one precondition and if the number of goals is bounded by a constant.

It is polynomial if each operator is restricted to no preconditions.

Additional boxes in the figure identify some commonalities.

The remainder of this paper is organized as follows. First, we describe previous results on the complexity of planning. Then, we define propositional planning. Next, we demonstrate our complexity results, and show how the blocks world is covered by one of our results. Finally, we discuss the impact of these results on the search for tractable planning.

[1] As is customary, we assume that PSPACE-complete problems are harder than NP-complete problems, which in turn are harder than polynomial problems. However, even P $\neq$ PSPACE is not yet proven.

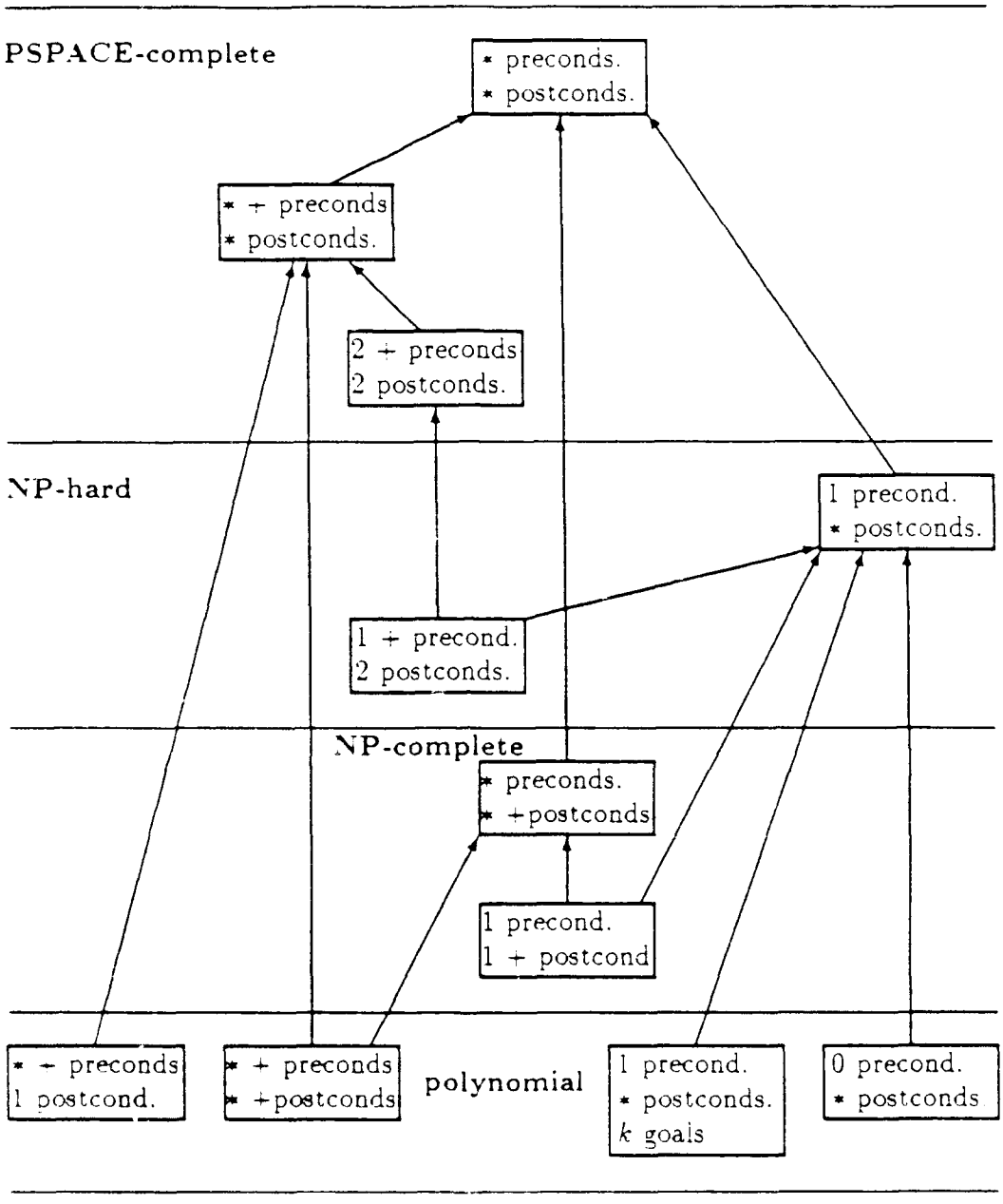


Figure 1: Complexity Results for Hierarchy of Propositional Planning Problems

2 Previous Results

The literature on planning is voluminous, and no attempt to properly survey the planning literature is attempted here. Instead, the reader is referred to Hendler et al. (1990). Despite the large literature, results on computational complexity are sparse. In turn, we discuss previous results from Dean and Boddy (1987), Korf (1987), and Chapman (1987).

Dean and Boddy (1987) analyze the problem of temporal projection—given a partial ordering of events and causal rules triggered by events, determine what conditions must be true after all of the events. Dean and Boddy's formalization of the temporal projection problem shares many features with planning, e.g., their causal rules contain antecedent conditions (preconditions) and added and deleted conditions (postconditions). In a sense, their events correspond to planning operators, especially if each event is associated with one causal rule. However, Dean and Boddy only consider problems of prediction in which a partial ordering of events is given, whereas the equivalent planning problem would be to find some ordering of events to achieve some set of conditions. Nevertheless, some of their results are similar to ours, e.g., temporal projection is intractable even if causal rules are limited to one antecedent condition, one added condition, and one deleted condition. Our notation for propositional planning is mostly borrowed from Dean and Boddy.

Korf (1987) considers how various global properties of planning problems (e.g., serializable subgoals, operator decomposability, abstraction) affect the complexity of using problem space search to find plans. In contrast, we focus exclusively on local properties of operators. However, except for Korf's own analysis of operator decomposability (Korf, 1985), neither he nor us describe the relationship from these properties of planning problems to the properties of operators. Clearly, this is a "gap" that future work should address.

Perhaps the most important complexity results for planning are due to Chapman's analysis of his planner TWEAK (Chapman, 1987). Because virtually all other planners are as expressive as TWEAK, his results have wide applicability. TWEAK includes the following representational features. The preconditions and postconditions of an operator schema are

finite sets of "propositions." A proposition is represented by a tuple of elements, which may be constants or variables, and can be negated. A postcondition of an operator can contain variables not specified by any precondition of the operator, which in effect allows creation of new constants. The initial state is a set of propositions. Each goal is also a proposition. A plan is a partial ordering on a set of operator instances; variables can be substituted with constants, but there is no requirement to do so.

Chapman proved the following the two theorems:

Theorem 1 (First Undecidability Theorem) *Any Turing machine with its input can be encoded as a planning problem in the* TWEAK *representation. Therefore, planning is undecidable, and no upper bound can be put on the amount of time required to solve a problem.*

Theorem 2 (Second Undecidability Theorem) *Planning is undecidable even with a finite initial state if the action representation is extended to represent actions whose effects are a function of their situation.*

For Theorem 1, Chapman's proof depends on an infinite initial state to represent an infinite Turing machine tape. For Theorem 2, allowing increment and decrement functions is sufficient for proving undecidability.

These theorems clearly demonstrate the difficulty of planning in general, but it is not obvious what features of TWEAK's representation are to blame for the complexity. What happens to the complexity, for example, if postconditions cannot introduce new variables? What happens if the size of states are bounded for any given instance of a planning problem? Are there any interesting restricted planning problems that are tractable? By considering a model of planning with considerably fewer features, our analysis begins to address these questions.

3 Propositional Planning

An instance of *propositional planning* is specified by a tuple $\langle \mathcal{P}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$, where:

$\mathcal{P}$ is a finite set of *conditions*;

$\mathcal{O}$ is a finite set of *operators*, where each operator has the form $\langle \varphi, \eta, \alpha, \delta \rangle$:

$\varphi \subseteq \mathcal{P}$ is a set of *positive preconditions*;
$\eta \subseteq \mathcal{P}$ is a set of *negative preconditions*;
$\alpha \subseteq \mathcal{P}$ is a set of *positive postconditions*;
$\delta \subseteq \mathcal{P}$ is a set of *negative postconditions*; and
$\varphi \cap \eta = \emptyset$ and $\alpha \cap \delta = \emptyset$.

$\mathcal{I} \subseteq \mathcal{P}$ is the *initial state*; and

$\mathcal{G} = \langle \mathcal{M}, \mathcal{N} \rangle$ is the *goal*:

$\mathcal{M} \subseteq \mathcal{P}$ is a set of *positive goals*;
$\mathcal{N} \subseteq \mathcal{P}$ is a set of *negative goals*; and
$\mathcal{M} \cap \mathcal{N} = \emptyset$.

That is, $\mathcal{P}$ is the set of conditions that are relevant. Any state can be specified by a subset $S \subseteq \mathcal{P}$, indicating that $p \in \mathcal{P}$ is true in that state if and only if $p \in S$. $\mathcal{O}$ is the set of the operators that can change one state to another. $\mathcal{I}$ specifies what conditions are true and false in the initial state, i.e., $p \in \mathcal{P}$ is initially true if $p \in \mathcal{I}$ and initially false otherwise. $\mathcal{G}$ specifies the goals, i.e., $S \subseteq \mathcal{P}$ is a *goal state* if and only if $\mathcal{M} \subseteq S$ and $S \cap \mathcal{N} = \emptyset$.

The effect of a finite sequence of operators $(o_1, o_2, \ldots, o_n)$ on a state S can be formalized as follows:

$$\begin{aligned}
&\mathit{Result}(S, (\,)) = S \\
&\mathit{Result}(S, (o)) = \begin{cases} (S \cup \alpha) \setminus \delta & \text{if } \varphi \subseteq S \wedge S \cap \eta = \emptyset \\ S & \text{otherwise} \end{cases} \\
&\mathit{Result}(S, (o_1, o_2, \ldots, o_n)) \\
&\qquad = \mathit{Result}(\mathit{Result}(S, (o_1)), (o_2, \ldots, o_n))
\end{aligned}$$

In essence, any operator can be applied to a state, but only has an effect if its preconditions are satisfied. If its preconditions are satisfied, its positive postconditions are added and its

negative postconditions are deleted, cf. Fikes and Nilsson, 1971). An operator can appear multiple times in a sequence of operators.

A finite sequence of operators $(o_1, o_2, \ldots, o_n)$ is a *solution* to an instance of propositional planning if $Result(\mathcal{I}, (o_1, o_2, \ldots, o_n))$ is a goal state.

An instance of a propositional planning problem is *satisfiable* if it has a solution. We define PLANSAT as the decision problem of determining whether an instance of propositional planning is satisfiable. Below, we consider the computational complexity of PLANSAT.

To show how a planning instance can be modeled by propositional planning, consider the Sussman anomaly. In this blocks-world instance, there are three blocks A, B, and C. Initially C is on A, A is on the table, and B is on the table. The goal is to have A on B, B on the table, and C on the table. Only one block at a time can be moved. The conditions, initial state, and goals can be represented as follows:

$$\begin{aligned}
\mathcal{P} &= \{A\text{-on-}B, A\text{-on-}C, B\text{-on-}A, B\text{-on-}C, C\text{-on-}A, C\text{-on-}B\} \\
\mathcal{I} &= \{C\text{-on-}A\} \\
\mathcal{M} &= \{A\text{-on-}B\} \\
\mathcal{N} &= \{B\text{-on-}A, B\text{-on-}C, C\text{-on-}A, C\text{-on-}B\}
\end{aligned}$$

The negative goals $\mathcal{N}$ encodes the fact that B and C are on the table if B and C are not on top of anything else. The operators will ensure that A-on-B and A-on-C cannot simultaneously hold.

The operator to unstack A can be represented as:

$$\begin{aligned}
\varphi &= \emptyset \\
\eta &= \{B\text{-on-}A, C\text{-on-}A\} \\
\alpha &= \emptyset \\
\delta &= \{A\text{-on-}B, A\text{-on-}C\}
\end{aligned}$$

That is, A can be moved to the table if nothing is on A. The result is that A will not be on top of any other block.

The operator to stack A on B can be represented as:

$$\begin{aligned}
\varphi &= \emptyset \\
\eta &= \{B\text{-on-}A, C\text{-on-}A, A\text{-on-}B, C\text{-on-}B\} \\
\alpha &= \{A\text{-on-}B\} \\
\delta &= \{A\text{-on-}C\}
\end{aligned}$$

That is, A can be stacked on B if nothing is on top of A or B. The result is that A will be on B and not on top of any other block.

Obviously, any blocks-world instance can be easily modeled as propositional planning. More generally, any TWEAK planning instance can be polynomially reduced to a propositional planning instance if each variable in an operator schema is limited to a polynomial number of values and if each operator schema is limited to a constant number of variables. An exponential/infinite number of values for a variable would lead to an exponential/infinite number of propositional planning conditions. A polynomial number of variables in an operator schema with more than one possible value for each variable would lead to at least an exponential number of propositional planning operators.

4 Complexity Results

We now describe and demonstrate our complexity results for propositional planning. As mentioned above, PLANSAT is the decision problem of determining whether an instance of propositional planning is satisfiable.

4.1 PSPACE-complete Propositional Planning

Theorem 3 *PLANSAT is PSPACE-complete.*

Proof: PLANSAT is in NPSPACE because a sequence of operators can be nondeterministically chosen, and the size of a state is bounded by the number of conditions. That is, if there are n conditions and there is a solution, then the length of the smallest solution path must be less than 2^n. Any solution of length 2^n or larger must have "loops," i.e., there must be some state that it visits twice. Such loops can be removed, resulting in a solution of length less than 2^n. Hence, no more than 2^n nondeterministic choices are required.

Because NPSPACE = PSPACE, PLANSAT is also in PSPACE.

Turing machines whose space is polynomially bounded can be polynomially reduced to PLANSAT. The PLANSAT conditions can be encoded (and roughly translated) as follows:

$p_{i,x}$	Tape position i contains symbol x.
$p_{q,i}$	The Turing machine is in state q and the input tape head is at the ith tape position.
$p_{q,i,x}$	The Turing machine was in state q and the input tape was at the ith position, which contained character x.
p_{accept}	The Turing machine accepts the input.

If q_0 is the initial state of the Turing machine, its input is $x_1 x_2 \ldots x_k$, and the space used by the Turing machine is bounded by n, then the the initial state and goals for propositional planning can be encoded as:

$$\begin{aligned}\mathcal{I} &= \{p_{q_0,0}, p_{0,\#}, p_{1,x_1}, p_{2,x_2}, \ldots, p_{k,x_k}, p_{k+1,\#}, p_{k+2,\#}, \ldots p_{n-1,\#}\} \\ \mathcal{M} &= \{p_{accept}\} \\ \mathcal{N} &= \emptyset\end{aligned}$$

$\mathcal{I}$ is encoded so that that position 1 to position k contain the input and the remaining positions (position 0 and positions $k+1$ to $n-1$) contain a special blank symbol $\#$

Suppose that the Turing machine is in state q, the input tape is at the ith position, x is the character at the ith position, and the transition is to replace x with y, move to the right, and be in state q'. This can be encoded using three operators:

$\varphi = \{p_{q,i}, p_{i,x}\}$	$\varphi = \{p_{q,i,x}, p_{i,x}\}$	$\varphi = \{p_{q,i,x}, p_{i,y}\}$
$\eta = \emptyset$	$\eta = \emptyset$	$\eta = \emptyset$
$\alpha = \{p_{q,i,x}\}$	$\alpha = \{p_{i,y}\}$	$\alpha = \{p_{q',i+1}\}$
$\delta = \{p_{q,i}\}$	$\delta = \{p_{i,x}\}$	$\delta = \{p_{q,i,x}\}$

The first operator "packs" all the information about the current position into a single condition. The second operator changes the symbol. The third operator moves to the next position and the new state. Boundary conditions can be easily handled by encoding no operators for $p_{q,-1}$ and $p_{q,n}$.

A Turing machine accepts an input if it is in an accepting state and no transition can be made from the current symbol. For each such case, an operator to add p_{accept} can be encoded.

Because there are a polynomial number of $\langle q, i, x \rangle$ combinations, there will be a polynomial number of conditions and operators. Thus, any PSPACE Turing machine with its input can be polynomially reduced to a propositional planning instance.

Note that none of the above operators requires more than two positive preconditions and two postconditions. This leads to the following corollary.

Corollary 4 *PLANSAT with operators restricted to two positive preconditions and two postconditions is PSPACE-complete.*

4.2 NP-complete and NP-hard Propositional Planning

Let PLANSAT+ be PLANSAT with operators restricted to positive postconditions. Note than any PLANSAT instance in which each condition is either never a positive postcondition or never a negative postcondition can be reduced to a PLANSAT+ instance.

Theorem 5 *PLANSAT+ is NP-complete.*

Proof: PLANSAT+ operators can never negate a condition, so the previous state will always be a subset of succeeding states. Also, operators within an operator sequence that have no effect can always be removed. Hence, if a solution exists, the length of the smallest solution can be no longer than the number of conditions. Thus, PLANSAT+ is in NP because only a polynomial number of nondeterministic choices is required.

3SAT can be polynomially reduced to PLANSAT+. 3SAT is the problem of satisfying a formula in propositional calculus in conjunctive normal form, in which each clause has at most three factors.

Let F be a formula in propositional calculus in 3SAT form. Let $U = \{u_1, u_2, \ldots, u_m\}$ be the variables used in F. Let n be the number of clauses in F. An equivalent PLANSAT+ instance can be constructed using the following types of conditions.

p_{u_i} u_i = true is selected.
$p_{\overline{u_i}}$ u_i = false is selected.
p_{c_j} The jth clause is satisfied.

The initial state and goals can be specified as:

$\mathcal{I} = \emptyset$
$\mathcal{M} = \{p_{c_1}, p_{c_2}, \ldots, p_{c_n}\}$
$\mathcal{N} = \emptyset$

That is, the goal is to satisfy each of the clauses.

For each variable u_i, two operators are needed:

$\varphi = \emptyset$	$\varphi = \emptyset$
$\eta = \{p_{\overline{u_i}}\}$	$\eta = \{p_{u_i}\}$
$\alpha = \{p_{u_i}\}$	$\alpha = \{p_{\overline{u_i}}\}$
$\delta = \emptyset$	$\delta = \emptyset$

That is, u_i = true can be selected only if u_i = false is not, and vice versa. In this fashion, only one of u_i = true and u_i = false can be selected.

For each case where a clause c_j contains a variable u_i, the first operator below is needed; for a negated variable $\overline{u_i}$, the second operator below is needed:

$\varphi = \{p_{u_i}\}$	$\varphi = \{p_{\overline{u_i}}\}$
$\eta = \emptyset$	$\eta = \emptyset$
$\alpha = \{p_{c_j}\}$	$\alpha = \{p_{c_j}\}$
$\delta = \emptyset$	$\delta = \emptyset$

Clearly, every p_{c_j} can be made true if and only if a satisfying assignment can be found. Thus PLANSAT+ is NP-hard. Since PLANSAT+ is also in NP, it follows that PLANSAT+ is NP-complete.

Note that each operator above only requires one precondition and one positive postcondition. This leads to the following corollary.

Corollary 6 *PLANSAT with operators restricted to one precondition and one positive postcondition is NP-complete.*

One additional result will complete the intractability results shown in Figure 1.

Theorem 7 *PLANSAT with operators restricted to one positive precondition and two postconditions is NP-hard.*

Proof: This can be shown by reduction from 3SAT, similar to that for Theorem 5. One additional type of condition is needed:

p_{x_i} No assignment to u_i has been selected.

The initial state is $\{p_{a_1}, p_{a_2}, \ldots, p_{a_m}\}$.

Now all that is needed are different operators for selecting an assignment.

$\rho = \{p_{a_i}\}$	$\rho = \{p_{a_i}\}$
$\eta = \emptyset$	$\eta = \emptyset$
$\alpha = \{p_{u_i}\}$	$\alpha = \{p_{\overline{u_i}}\}$
$\delta = \{p_{a_i}\}$	$\delta = \{p_{a_i}\}$

Again, both u_i = true and u_i = false cannot be selected.

The same operators for clauses as in the proof for Theorem 5 can be used. A "dummy" postcondition can be used to increase the number of postconditions to two.

4.3 Polynomial Propositional Planning

Theorem 8 *PLANSAT with operators restricted to positive preconditions and one postcondition is polynomial.*

Proof: The difficulty is that some negative goals might need to be temporarily positive to make some positive goals positive or some negative goals negative. Fortunately, because of the restrictions on the operators, it is sufficient to only consider plans that first make conditions positive and then make conditions negative. Consider a sequence of operators $(o_1, o_2, \ldots, o_n)$ in which the preconditions of each operator become true. Suppose that there are adjacent operators o_i and o_{i+1} such that o_i makes a condition negative and o_{i+1} makes a condition positive. Let p_i be o_i's negative postcondition and p_{i+1} be o_{i+1}'s positive postcondition. If $p_i = p_{i+1}$, then o_i can be deleted because o_{i+1} reverses o_i's effect. If $p_i \neq p_{i+1}$, then o_i can be switched with o_{i+1} because their postconditions are independent of each other's preconditions. Leaving p_i positive cannot cause o_{i+1}'s preconditions to become false, and making p_{i+1} positive cannot cause o_i's preconditions to become false. Repeating these changes until there no such adjacent operators will result in an equivalent sequence of operators that first makes conditions positive and then makes conditions negative.

Let *Turnon* then be a subroutine that inputs a subset of conditions $X \subseteq \mathcal{P}$. *Turnon* determines a maximal state $S \subseteq \mathcal{P} \setminus X$ that can be reached from the initial state using operators with positive preconditions, i.e., result in as many positive conditions as possible

while keeping all conditions in X negative. Because all preconditions are positive, *Turnon* is polynomial. It is sufficient to keep checking all the operators with positive postconditions until no more conditions can be made positive. It can be shown that there is exactly one such maximal state.

Let *Turnoff* be another subroutine that inputs a subset of conditions $S \subseteq \mathcal{P}$. *Turnoff* determines a maximal state $S' \subseteq S$ such that $S \setminus \mathcal{N}$ can be reached from S' using operators with negative preconditions. *Turnoff* is polynomial because it is possible to work backward from $S \setminus \mathcal{N}$ to determine what conditions in $S \cap \mathcal{N}$ could have been positive. It can be shown that there is exactly one such maximal state.

The following algorithm determines if a solution exists by iterating between *Turnon* and *Turnoff*:

$X \leftarrow \emptyset$
loop
  $S \leftarrow$ *Turnon*(X)
  if $\mathcal{M} \not\subseteq S$ then reject
  $S' \leftarrow$ *Turnoff*(S)
  if $S = S'$ then accept
  $X \leftarrow X \cup (S \setminus S')$
  if $X \cap \mathcal{I} \neq \emptyset$ then reject
end loop

This algorithm is polynomial because *Turnon* and *Turnoff* are polynomial, and X grows monotonically. The algorithm works because every element of X is a negative goal, that, if initially positive or made positive, prevents the goal state from being reached.

Theorem 9 *PLANSAT with operators restricted to positive preconditions and positive postconditions is polynomial.*

Proof: The algorithm is trivial. Simply apply operators that do not make any negative goals positive until no more such operators have any effect. If a goal state is reached, then accept, otherwise reject.

Theorem 10 *PLANSAT with k goals and operators restricted to one precondition is polynomial.*

Proof: It is possible to work forward from the initial state. Construct all possible combinations of k conditions. Mark those combinations true of the initial state. For each marked combination, mark any combinations that can be reached from that combination via an operator. After all possible combinations are marked, if the combination of conditions corresponding to the goal is marked then accept, otherwise reject.

This is why the algorithm works. Consider a solution plan and any one of the k goals. To reach this goal, there must be a "chain" of operators leading from one condition in the initial state through one condition at a time until the goal is reached. Consider now the k chains of operators for the k goals. Consider also any state reached during the execution of the solution plan. This state will correspond to k nodes on the k chains. Any state that satisfies the k conditions corresponding to those nodes can reach the goal state. Since this is true for all states reached by the solution plan, it must be the case that only k conditions at a time need to be considered to determine what combinations of k conditions can be reached.

Theorem 11 *PLANSAT with operators restricted to no preconditions is polynomial.*

Proof: It is possible to work backwards from the goals. First look for operators that do not clobber any of the goals. Goals that are achieved by these operators can be removed from consideration. These operators can also be removed from consideration. Then look for operators that do not clobber the remaining goals, and remove from consideration these operators and the goals they achieve. This can be repeated until the remaining goals are true of the initial state (accept) or until no appropriate operators can be found (reject).

4.4 The Blocks World

Theorem 8 can be used to show that the blocks-world problem is tractable.

Theorem 12 *The blocks-world problem can be solved using operators restricted to positive preconditions and one postcondition.*

Proof: We note that stacking one block on another can be accomplished by first moving the former block on the table and then moving it on top of the latter block. Thus, solving any

blocks-world instance only requires operators to move a block to the table and to move a block from the table.

Let $\{B_1, B_2, \ldots, B_n\}$ be the blocks in an instance of the blocks-world problem. The conditions can be encoded as follows:

$p_{i,j}$ B_i is *not* on top of B_j.

If B_i is on the table, then all $p_{i,k}$ will be true. If B_i has a clear top, then all $p_{k,i}$ will be true. If B_i is on top of B_j, then all $p_{i,k}$ except for $p_{i,j}$ will be true.

For each B_i and B_j, $i \neq j$, the operator to move B_i from on top of B_j to the table can be encoded as:

$$\begin{aligned}\varphi &= \{p_{1,i}, p_{2,i}, \ldots, p_{n,i}, p_{1,j}, p_{2,j}, \ldots, p_{i-1,j}, p_{i+1,j}, \ldots, p_{n,j}\}\\ \eta &= \emptyset\\ \alpha &= \{p_{i,j}\}\\ \delta &= \emptyset\end{aligned}$$

That is, if nothing is on B_i and nothing is on B_j except possibly B_i, then when this operator is applied, the result is that B_i will not be on top of B_j.

For each B_i and B_j, $i \neq j$, the operator to move B_i from on the table to on top of B_j can be encoded as:

$$\begin{aligned}\varphi &= \{p_{1,i}, p_{2,i}, \ldots, p_{n,i}, p_{1,j}, p_{2,j}, \ldots, p_{n,j}\}\\ \eta &= \emptyset\\ \alpha &= \emptyset\\ \delta &= \{p_{i,j}\}\end{aligned}$$

That is, if nothing is on B_i and B_j, then when this operator is applied, the result is that B_i will be on top of B_j.

Since there are only $O(n^2)$ $\langle i, j\rangle$ combinations, only $O(n^2)$ conditions and operators are needed to encode a blocks-world instance.

As required, all preconditions are positive and each operator has only one postcondition. Thus, Theorem 8, in a sense, explains why the blocks world is tractable.[2]

[2]The algorithm for Theorem 8 corresponds to the unimaginative, but robust, strategy of moving all the blocks to the table, which makes all the conditions positive, and then forming the stacks from the table on up.

5 Remarks

Planning is intractable even if the size of states are bounded and operators have no variables. Merely allowing two preconditions and two postconditions for operators gives rise to an extremely hard problem. However, operators must have preconditions, postconditions, and apparently many more "features" to implement any interesting domain (Chapman, 1987; Hendler et al., 1990). While additional features might be good for making a planner more useful as a programming tool, generality has its downside—tractability cannot be guaranteed unless there are sufficient restrictions on the operators.

In the context of propositional planning, we have discovered four restricted planning problems that are tractable. Restricting operators to positive preconditions and one postcondition explains the tractability of the blocks-world. Restricting operators to positive preconditions and positive postconditions, though unrealistic, makes the list of tractable problems more complete. Restricting operators to one precondition and limiting the number of goals is the only case where restricting the number of goals leads to tractability. Restricting operators to no preconditions shows that planning can be done efficiently if preconditions can be ignored, e.g., if preconditions of operators can be easily satisfied without clobbering already achieved goals.

However, we expect that many, if not most, planning domains violate these categories. Thus, these domains cannot be shown to be tractable based on restrictions on the local properties of operators. As mentioned in the section on previous research, Korf (1987) lists several global properties of planning problems that lead to efficient search for plans. Understanding how these properties are realized as restrictions on the set of operators as a whole is a promising research approach.

Unfortunately, the prospects for a single domain-independent planning algorithm are pessimistic. The four tractable problems above appear to require quite different algorithms, and many other tractable planning problems are yet to be discovered. We believe that it will be more fruitful to adopt different methods for different types of planning problems.

References

Chapman, D. (1987). Planning for conjunctive goals. *Artificial Intelligence*, 32(3):333–377.

Dean, T. and Boddy, M. (1987). Reasoning about partially ordered events. *Artificial Intelligence*, 36(3):375–399.

Fikes, R. E. and Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3/4):189–208.

Hendler, J., Tate, A., and Drummond, M. (1990). AI planning: Systems and techniques. *AI Magazine*, 11(2):61–77.

Korf, R. E. (1985). Macro-operators: A weak method for learning. *Artificial Intelligence*, 26(1):35–77.

Korf, R. E. (1987). Planning as search: A quantitative approach. *Artificial Intelligence*, 33(1):65–88.

Generic Tasks and Soar

Todd R. Johnson, Jack W. Smith, Jr., B. Chandrasekaran

Laboratory for Artificial Intelligence Research
Department of Computer and Information Science
The Ohio State University
Columbus Ohio, 43210

Abstract

This paper describes our research which is an attempt to retain as many of the advantages as possible of both task-specific architectures and the flexibility and generality of more general problem-solving architectures like Soar. It investigates how task-specific architectures can be constructed in the Soar framework and integrated and used in a flexible manner. The results of our investigation are a preliminary step towards unification of general and task-specific problem solving theories and architectures.

Introduction

Two trends can be discerned in research in problem solving architectures in the last few years: On one hand, interest in *task-specific architectures* [Clancey, 1985, Marcus and McDermott, 1987, Chandrasekaran, 1986] has grown, wherein types of problems of general utility are identified, and special architectures that support the development of problem solving systems for those types of problems are proposed. These architectures help in the acquisition and specification of knowledge by providing inference methods that are appropriate for the type of problem. However, knowledge-based systems which use only one type of problem solving method are very brittle, and adding more types of methods requires a principled approach to integrating them in a flexible way

Contrasting with this trend is the proposal for a flexible, general architecture contained in the work on Soar [Laird et al., 1987]. Soar has features which make it attractive for flexible use of all potentially relevant knowledge or methods. But as a theory Soar does not make commitments to specific types of problem solvers or provide guidance for their construction.

In this paper we investigate how task-specific architectures can be constructed in Soar to retain as many of the advantages as possible of both approaches. We will be using examples from the Generic Task (GT) approach for building knowledge-based systems in our discussion since this approach had its genesis at our Laboratory where it has further been developed and applied for a number of problems; however the ideas are applicable to other task-specific approaches as well.

The GT Paradigm

The GT paradigm is a theory of types of goals and the problem solving methods needed to achieve each type. By problem solving method we mean the specification of behavior to achieve a goal. The paradigm has three main parts:

1. The problem solving of an intelligent agent can be characterized by generic types of goals. Many problems can be solved using some combination of these types.
2. For each type of goal there are one or more problem solving methods, any one of which can potentially be used to achieve the goal.
3. Each problem solving method requires certain kinds of knowledge of the task in order to execute. These are called the *operational demands* of the method [Laird et al., 1986].

The term *generic task* refers to the combination of a type of goal with a problem solving method and the kinds of knowledge needed to use the method. The GT for classification by establish-refine (called the E-R GT) is given as:

Type of Goal Classify a (possibly complex) description of a situation as a class in a set of categories. An instance of this goal is the classification of a medical case description as one of a set of diseases.

Problem Solving Method This is a hierarchical classification method that works by creating and testing hypotheses about the plausibility that the description of the situation represents an instance of one or more of the classes.

1. If there are no initial hypotheses about what class the description is an instance of, then try to suggest at least one.
2. Try to confirm or reject any hypothesis that is suggested.
3. If a hypothesis is confirmed, determine the possible refinements of the hypothesis and suggest them.
4. If the goal is not met, go to step 2.

Kinds of Knowledge These consist of a refinement hierarchy, hypotheses about the presence of classes, confirmation/rule-out knowledge for these hypotheses, and knowledge to determine when the goal of classification has been achieved.

In addition to classification by establish-refine, GT's have been created for pattern directed hypothesis matching [Johnson et al., 1988], object synthesis by plan selection and refinement [Brown and Chandrasekaran, 1985], and assembly of explanatory hypotheses [Josephson et al., 1987].

GT Systems

A specialized architecture or shell has been constructed for each GT. Each architecture is a combination of an inference engine with a knowledge base. The inference engine is a procedural representation of a GT's problem solving method. The knowledge base provides primitives for encoding the domain specific knowledge needed to instantiate the procedure. We refer to the combination of the encoding of the domain knowledge in the knowledge base and the method that can use it as a problem-solver.

This system building approach offers a number of advantages: First, it is easy to decide when a GT architecture can be used because the knowledge operationally demanded by the method is explicit in the definition of the GT. Second, knowledge acquisition is facilitated because the representational primitives of the knowledge base directly correspond to the kinds of domain knowledge that must be gathered. Third, explanation based on a run-time trace can be couched in terms of the method and knowledge being used to apply it.

Problems with GT Systems

Many flexibility problems arise because a GT architecture contains assumptions not present in the original GT problem solving method. For example, our architecture for hierarchical classification assumes that hypotheses are generated from a pre-defined hierarchy. While this is a common way to generate refinements, other ways exist and might be useful in certain domains. Second, the architecture immediately generates refinements for a confirmed hypothesis. An alternative is to test all the hypotheses in the current state before refining any that were confirmed. Third, the architecture assumes that any problem solver it calls will correctly function. We cannot easily modify the architecture to gracefully handle these situations.

Another set of problems involves the integration of multiple GT problem solvers. The simplest kind of integration is when one problem solver calls on another as a direct means to achieve a subgoal. This is easily done using our current architectures by directly invoking the method and domain knowledge needed to achieve a subgoal. However, sometimes we require more interaction between the problem solvers. For example, in our medical diagnosis systems the hypothesis assembly problem solver has knowledge about those diseases that can occur together and those that are mutually exclusive. This knowledge can be used help guide the classification of diseases; however, it is difficult to implement because the classification architecture has no place for representing or using this knowledge. Our only solution was to specially modify both architectures so that they could interact in the desired way.

Finally, new methods are difficult to add to existing problem solvers; each problem solver must be modified to recognize and use a new method. We would like to have the system automatically consider methods based on the type of goal a method is designed to achieve.

How can Soar Help?

In Soar, all problem solving is viewed as search for a goal state in a problem space. Operators are used to move from state to state. Knowledge in the form of productions is used to select problem spaces, states, and operators. Productions generate *preferences* for an object (ie. a problem space, state, or operator) that indicate how the object relates to the current situation or other objects. Whenever the directly available knowledge is insufficient to make progress Soar automatically generates a subgoal. Therefore, every decision that needs to be made can become a goal to be achieved by search-

ing a problem space. This is called *universal subgoaling*. In knowledge lean situations Soar can make progress by using an appropriate weak method. The weak methods are not explicitly programmed in Soar, but arise from the knowledge available to solve a problem. If the processing in a subgoal is no longer needed, Soar will automatically terminate the subgoal and resume problem solving in a higher level goal. This is called *automatic goal termination*.

Each of these features directly relate to one or more of the limitations with GT systems. The selection of alternatives via preferences allows new options and knowledge to be easily added to existing systems. Brittleness is decreased because of Soar's ability to automatically create subgoals to overcome failures and its ability to fall back on weak methods. Finally, automatic goal termination eliminates unnecessary computation and provides a more natural flow of control.

Mapping GT's to Soar

We have begun to map GT's to the Soar architecture in a straight-forward manner. Each GT is implemented as a problem space; the states represent the developing solution and the operators and operator suggestion rules implement the problem solving method. The required kinds of knowledge can either be represented directly by productions or generated at run-time using additional problem spaces.

To illustrate, we present ER-Soar, an implementation of the E-R GT in Soar. We use a single problem space with three operators: suggest-initial-hypotheses, establish, and generate-refinements.

State Representation The state contains those hypotheses that have been considered and those that are worth immediately considering. Any hypotheses in the state that are refinements of other hypotheses (also in the state) are linked together to form a refinement hierarchy. Each hypothesis also has an indication of whether it has been confirmed, rejected, or not yet judged, and whether it has been refined or not.

Operators The classify problem-space uses 3 operators:

suggest-initial-hypotheses Determine one or more initial hypotheses.

establish <hyp> Determine whether the hypothesis, <hyp>, should be confirmed or rejected.

generate-refinements <hyp> Generate and add to the state those hypotheses that should be considered as a refinement of <hyp>.

Operator Instantiation An establish operator is created for any hypothesis in the current state that has not yet been judged. A generate-refinements operator is created for any hypothesis that is confirmed but not refined. A suggest-initial-hypotheses operator is created if there are no hypotheses in the current state.

Domain knowledge To use the E-R strategy in a particular domain, knowledge to perform the following functions must be added to the Soar implementation.

- Create the initial state containing one or more initial hypotheses.
- Detect when classification is complete.
- Implement the three operators.

Operator Implementation There are many ways to implement the operators used in the classify space. To make ER-Soar easy to use we have implemented a method for generating refinements from a pre-defined hierarchy and a method for establishing hypotheses based on a confidence value.

Discussion

ER-Soar combines the advantages of the GT approach with the advantages of the Soar architecture. Knowledge acquisition, ease of use, and explanation are all facilitated in ER-Soar because subgoals of the problem solving method and the kinds of knowledge needed to use the method are explicitly represented in the implementation. The subgoals of the method are directly represented as problem space operators. The kinds of knowledge needed to use the method are either encoded in productions or computed in a subgoal. The same advantages apply to the supplied methods for achieving subgoals. Finally, the implementation mirrors the GT specification quite closely making ER-Soar easy to understand and use.

ER-Soar overcomes many of the problems suffered by previous GT systems. Automatic subgoaling allows unanticipated situations to be detected and handled. If no specific method for handling the situation is available, an appropriate weak method can be used. Whenever a goal needs to be achieved it is done by first suggesting problem-spaces and then selecting one to use. This allows new methods in the form of problem-spaces to be

easily added to existing problem solvers. If no specific technique exists to determine which method to use, Soar will try to pick one using a weak method. Automatic goal termination provides an integration functionality not available in previous GT architectures. In general, the integration capabilities of ER-Soar are greatly enhanced. Because of preferences and the additive nature of productions, new knowledge can be added to integrate ER-Soar with other methods without modifying existing control knowledge.

Conclusion

ER-Soar illustrates how the advantages of task-specific architectures can be combined with the advantages of a general architecture. The approach used to create ER-Soar is simple and can easily be applied to other task-specific architectures. We are currently using this approach to create Soar versions of the GT's for hypothesis matching and hypothesis assembly. Following this, we will investigate various ways of integrating the three methods.

Acknowledgements

This research is supported by National Heart Lung and Blood Institute grant HL-38776, Air Force Office of Scientific Research grant 82-0255, and Defense Advanced Research Projects Agency, RADC Contract F30602-85-C-0010. Dr. Jack W. Smith, Jr. is supported by National Library of Medicine Career Development Award LM-00083. Computer facilities were enhanced by gifts from Xerox Corporation.

References

[Brown and Chandrasekaran, 1985] Brown, D. C. and Chandrasekaran, B. (1985). Plan selection in design problem-solving. In *Proc. of AISB85 Conference*, Warwick, England. The Society for AI and the Simulation of Behavior (AISB).

[Chandrasekaran, 1986] Chandrasekaran, B. (1986). Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design. *IEEE Expert*, 1(3):23–30.

[Clancey, 1985] Clancey, W. J. (1985). Heuristic classification. *Artificial Intelligence*, 27(3):289–350.

[Johnson et al., 1988] Johnson, T. R., Smith, Jr., J., and Bylander, T. (1988). HYPER: Hypothesis matching using compiled knowledge. Technical report, Lab. for AI Research, CIS Dept., The Ohio State University, Columbus, Ohio.

[Josephson et al., 1987] Josephson, J. R., Chandrasekaran, B., Smith, J. W., and Tanner, M. C. (1987) A mechanism for forming composite explanatory hypotheses. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-17(3):445–454.

[Laird et al., 1986] Laird, J., Rosenbloom, P., and Newell, A. (1986). *Universal Subgoaling and Chunking*. Kluwer Academic Publishers, Massachusets.

[Laird et al., 1987] Laird, J. E., Newell, A., and Rosenbloom, P. S. (1987). SOAR: An architecture for general intelligence. *Artificial Intelligence*, 33:1–64.

[Marcus and McDermott, 1987] Marcus, S. and McDermott, J. (1987). Salt: A knowledge acquisition tool for propose and revise systems. Technical report, Carnegie-Mellon University.

This is to appear as part of "Knowledge Compilation: A Symposium", A. Goel (Guest Editor), IEEE Expert.

Models vs Rules, Deep vs Compiled, Content vs Form: Some Distinctions in Knowledge Systems Research

B. Chandrasekaran

October 2, 1990. Revised January 7, 1991

1 Goal of Contribution

In this contribution, I want to discuss a number of distinctions that have been made in the literature on knowledge system design that require further careful analysis: model-based vs rule-based problem solving, models vs experiential or associational knowledge, deep vs compiled knowledge, and content vs form. I will focus on the diagnostic task in my discussion, since these distinctions have most often been debated in the context of this problem, but the issues are of relevance to knowledge systems in general.

2 Models and Rules

The "model-based vs rule-based" distinction mixes categories and hence does not set up genuine alternatives. All descriptions of any aspect of reality are *models*. Thus Mycin [1], MDX [2], Neomycin [3], MDX-2 [4] are *all* model-based systems for diagnosis, i.e., they all use bodies of knowledge that describe their domain, but the vocabularies in which the models are described in these systems are different. The concepts used in the description of these models ought to be distinguished from their symbol-level representations [5]. "Rules" are one symbol-level commitment to *represent* the models.

Almost invariably, the term "model" has been used to refer one particular type of model: one which is intended for the task of diagnosis, and in which

the device under diagnosis is described in terms of components, relations between components, and some sort of a behavioral description of the components. In the rest of the discussion I want to call this class of models *SBF models*, to indicate that their content deals with structure/behavior/function of the components of the device.

Three distinct senses of the phrase *rule-based* were rolled into one in early discussions in the field of knowledge-based systems.

- i. *Rule systems as general-purpose programming languages.* Computer science has generated a family of rule-based programming languages (e.g., Post Production Systems, Markov Algorithms). The OPS family and Emycin were additional entrants to this family of symbol-level proposals. These rule languages are especially useful to support a declarative style of knowledge representation. Many expert systems built using rules were simply algorithms implemented using a rule-programming language, without a commitment to any specific view about intelligence, human or artificial, or even knowledge.

- ii. *Rule systems as human cognitive architecture.* Newell and Simon [6] proposed a type of rule-based architecture as a model for human cognition. This is a model at the computational level, that is, it lays out the data structures and operations by which computation proceeds. Rule-based systems in AI derived some of their aura from this implied connection to cognitive architectures. However, to take a well-known system as an example, Mycin owes its success less to the use of the rule system as a cognitive model, and more to its use as a knowledge programming language to implement a classification strategy.

- iii. *Rules as "rules of thumb," heuristic associations.* The rule-based expert system movement also had a theory of expertise, that it was in fact a collection of heuristic associations. Mycin rules were supposedly of this type. It was additionally proposed often that the origin of the heuristic associations was empirical problem solving experiece. Expertise of this type was contrasted to so-called theory-based knowledge, which was supposed to be based on scientific principles underlying the domain, but not yet in a form useful for problem solving. Use of rules in this sense is really a statement about the origin and character of

the knowledge represented, not about rule systems as programming languages or cognitive architectures.

There is quite a potential for confusion when all these senses of rules are not distinguished. In discussions on the model-based versus rule-based distinction, for example, it is generally not made clear when rules are used as symbol-level representational ideas and when as empirical associations.

Because of the equation of rules with empirical associations, suggestions began to be made that perhaps we need "models" as opposed to "rules," where models were supposed to stand for "first principles," which were then equated with "scientific laws." Another sequence of conflations came about. Thus, in parallel with identifying rules with empirical associations, there was this association of "models" with physics or first principles.

Consider a piece of knowledge that a diagnostician might use, "excessive temperature at the outlet valve indicates that Valve A might be blocked." He might actually have derived it by generating an account of the consequences of the broken valve, or he might have stumbled on it by association. Thus the fact that the diagnostician has this knowledge and uses it for diagnosis does not say anything about the origin of this knowledge, whether it was obtained associationally, experientially or whatever. Conversely, representing $E = MC^2$ as a rule does not suddenly make it "experiential" or "associational." This common, but erroneous, identification of knowledge in rule form with its origin as experiential or associational is a conflation of the role or origin of knowledge with its *symbol level representation*. Equating expertise with rules and in turn with empirical association continues to be a source of confusion.

The diagnostic task requires knowledge that helps to map from observations to diagnostic hypotheses. The piece of knowledge relating a certain temperature to a valve blockage is thus directly usable in the diagnostic task. It relates an observation to a malfunction hypothesis directly, and during diagnostic problem solving the causal pathway by means of which the blockage causes the high temperature need not be generated or accessed. This points to the importance of understand-

ing the relation between tasks, methods and types of knowledge. In [7], I discussed the nature of task-specific knowledge for diagnosis and how that knowledge might be related to domain knowledge in other forms. Are there other sources of knowledge from which knowledge in a form directly usable for diagnosis can be derived, or "compiled"? I suggested that such a knowledge source is likely to contain knowledge of the structure of the system that is being diagnosed. That paper also made the connection between tasks and types of knowledge needed for accomplishing them, and thus suggested that the idea of compilation was closely connected with the abstract task to be accomplished.

3 The Diagnostic Task

For the purpose of this discussion, we can identify a restricted class of diagnostic problems as follows. I need to emphasize the word "restricted," since many important diagnostic problems are not covered by the definition that follows. We have a device which can be described as a set of components connected in a certain fashion. Information, energy or physical fluids may flow through the device in a manner determined by the behavioral properties of the components and the connections. The components might be recursively defined as devices themselves down to some level that is convenient or sufficient for the purpose at hand. A set S of state variables is defined. (To avoid cumbersomeness, I will use the term "state" in further discussion, when I really mean a partial state, often described by using a state variable.) A subset O of S is directly observable and yield *observations*. A subset O_f of O is defined such that for observations in this set, expected or normal values are known or can be inferred immediately. Deviations from the expected values of observations trigger the diagnostic process.

A class of diagnostic problems for the device is specified by O, O_f along with specification of expected values for observations in O_f, and a set C of *causes* of interest. For the restricted diagnostic problem under discussion, causes are normally components that malfunction, relations between components that are out of specifications, or input information or fluids that are out of specification.

An instance of the diagnostic problem in this class occurs when a set of observations, some of which deviate from expected values, is given. The goal is to determine the subset of C that is the most likely to have caused the observations.

Note that O is in general a *small* subset of the set of all state variables of the device. We need to reason about other, unobservable, states from the observations.

4 Compiled and Deep Knowledge

4.1 Tasks, Methods and Knowledge

For concreteness of discussion, I will use the language of search in problem spaces to discuss tasks and methods. Similar points can be made for viewing problem solving as inference-making, constraint-satisfaction, best-match retrieval, or other alternative formulations. Different tasks/methods need different kinds of knowledge. If the knowledge needed for for a task/method is directly available, we will say that the system has compiled knowledge.

Task: A problem solving task, or simply a task, is defined by characterizing the starting and desired end states of knowledge of the problem solving process.

For the diagnostic problem described above, the task can be stated as:

Starting state: includes a set of observations of the system under diagnosis, where some of the observations deviate from the expected;

Desired end state: includes a subset of causes (e.g., malfunctioning components or incorrect relations between components) which together caused the observations.

Method: A method is a proposal about how to organize information processing to achieve the goal. A method can be thought of as a collection of problem spaces that are made available to achieve the task. A method can be described in terms of the spaces of alternative partial solutions, the operators in each space that help generate candidate solutions, the objects that the operators operate on, and any additional

knowledge about how to organize operator application to achieve the task. There may be many methods available to achieve a given task.

A common and simple method for the above version of the diagnostic task, applicable when the malfunction causes do not interact, is *classification*. In this method, the observations are mapped onto the set of malfunction causes C. Some sort of likelihood of each element in C is computed, and hypotheses whose likelihoods are over a threshold are proposed as the answer. Two distinct types of problem solving go on. In one, the space of malfunctions is explored. Determining the likelihood of, and the explanatory coverage by, each malfunction constitutes the other type of problem solving.

Knowledge for the Method: In order to apply method M for task T, the agent needs knowledge of certain types. This knowledge, say K(T,M), is used to set up the problem spaces and apply the operators to change knowledge states.

For the classification method for the version of the diagnostic task that I have been discussing, we need the following types of knowledge:

-The set C of malfunction hypotheses.

-How observations and malfunction hypotheses are related, so that likelihoods of various causes can be computed.

4.2 Depth of Knowledge Is Relative

If the knowledge K(T,M) needed by method M for task T is directly available to a knowledge system, we say that the knowledge system has the needed knowledge available in a *compiled* form. Here the word "compiled" is being used in the ordinary English sense of "collected together."

If knowledge element k in K(T,M) can be derived from another type of knowledge K' by additional reasoning or problem solving activities, then K' is *deep relative to* k. Knowledge elements of K' itself might be derivable from yet another body of knowledge K" by additional problem solving. Whether or not to retain the knowledge element k

or to discard it and regenerate it as needed is a separate issue and discussed elsewhere in this paper.

For the diagnostic example, consider a knowledge element k(h,o), defined as knowledge that indicates if an observation o could have been caused by a malfunction hypothesis h in C. Most first generation diagnostic systems used some version of the classificatory method and their knowledge bases had this information in some form in their knowledge bases. For example, Mycin had rules which connected observations and hypotheses with degrees of certainty. If k(h,o) is not directly available, it might be derivable from other types of knowledge. I will consider two examples: causal sequences of states and SBF models of components.

Causal account of malfunction h. Consider knowledge in the form of a causal account of malfunction h, that is, a sequence of states of interest linking a state and its causal consequence as the malfunction h propagates. Some of the states in the causal sequence might be observables. The causal account can have branches conditioned on other device parameters and inputs. This causal account can be traced to see if it includes observational states of interest. The functional representation [5] organizes causal sequences by explicitly indicating the roles played by components and domain laws. These causal accounts of malfunctions (or functions, for that matter) can themselves be derived from other sources of knowledge.

Structure-Function Description of Components. Consider what I have labeled an SBF model for a device, that is, one which contains a description of components of the device, the input/output specifications of the components, and composition knowledge that enables one to derive input/output properties of various configurations of components. If malfunction h is a specific component violating its input-output specifications in a particular way, then a causal account (or parts of it) might be derived by using simulation methods that use the SBF specification of components, and k(h,o) can be derived from this causal sequence. As I said earlier, use of SBF models in this manner for diagnosis has been called, somewhat imprecisely, model-based reasoning in diagnosis.

It is worth restating that the notion of depth of knowledge is relative, and there is no one body of knowledge that can be said to be deep

relative to a knowledge element k. In the above example, causal accounts and SBF models are both deep relative to k(h,o). A somewhat different definition of depth is given in [9], but, it can be argued that, at least for the case of device diagnosis, the underlying senses of depth converge.

A brief comparison with an aspect of SOAR [10] is relevant. When SOAR is missing knowledge needed to apply an operator in a problem space, it sets up a new problem space in which the needed knowledge may be generated. In the diagnosis example, the diagnostic hypothesis space is the original problem space, k(h,o) is part of the knowledge needed to esablish or reject a hypothesis h, and the causal knowledge structures are candidates for additional problem spaces to generate this knowledge.

4.3 Deep Models are More Robust

Suppose a piece of knowledge k can be derived from another piece of knowledge k' and vice versa. In this case, k and k' are deep models relative to each other, but that is not very interesting. In general, however, there is a loss of information when one compiles a knowledge element from a deep model. Except in trivial problem spaces, discarding those portions of the space which do not help in the achievement of the current goal involves loss of information. In my examples, the causal account of malfunction contains information about internal device states, which is missing in k(h,o). SBF models contain information about normal functioning of different components which is missing in the causal accounts of malfunctions. It is this uni-directionality of information retention that makes some knowledge types deep with respect to others.

Deep models are more robust in the sense that they are potentially applicable for a larger variety of problems than knowledge that is tuned for a task or a particular version of it. Because of changes in domain technology, e.g., difference in probes, suppose some internal states which were not observable have been made observable and vice versa. The device is the same, the malfunctions are still the same, but the diagnostic task is now different. K(T,M) needed for this new version of

the task is correspondingly different. However, the causal accounts of malfunctions can be re-processed to obtain k(h,o)'s for building up the new K(T,M).

Suppose the device remains the same, but the purposes for which it is being (i.e., its functions) used have changed. The set of malfunctions is no longer the same. Again, there is a change in the diagnostic task, and consequently K(T,M) is different as well. The causal accounts that had been generated for the previous set of malfunctions are no longer useful. However, the SBF models of the components and the simulation process can still be used to derive the causal accounts for the new malfunctions, and consequently to derive the elements of the new K(T,M).

4.4 Other Senses of Compilation

Compilation has been an especially difficult term, since it has technical connotations in computer science as well as common linguistic use. My sense of compilation as generating knowledge required to support a method for a task is related to the common meaning of "gathering" or "collecting." In computer science usage in the context of programming languages, compilation is a form of translation, and no loss of information is implied. Also, compilation is often contrasted to "interpretation," a run-time translation of only a local portion of the program. My use of the term compilation does not involve these senses and implications.

4.5 When to Build Compiled Knowledge Structures?

Whether knowledge needed for a task is generated at run-time on an as-needed basis by accessing only portions of the deep knowledge structure that are relevant to the current problem, or a large store of such task-specific pieces is built a priori and made available, is an issue that is separate from the idea that tasks and methods determine the type of

knowledge needed, which in turn can be derived from more general knowledge structures.

Nevertheless, when to compile and store a knowledge element is an important question. This is really an instance of the more general question of when to "cache" the result of any problem solving. Three criteria spring to mind for caching:

- The problem solving activity in the generation of the knowledge element involved an exponential amount of computation, e.g., exponential search.
- The knowledge element generated is likely to be needed sufficiently often.
- The knowledge element can be generalized and indexed properly by the situations for which it is appropriate.

As I mentioned, a good vocabulary of tasks, methods and the knowledge needed to support them eases the indexing problem. In the remainder of this section, I want to investigate the nature of search in compilation. I will continue to use the diagnostic task for my discussion.

In Section 4.1, I identified two types of knowledge needed for the classification method: malfunction hypotheses and knowledge of relations between observations and hypotheses. SBF models can help generate both these types of knowledge. However, in what follows, I confine my attention to just one specific part of the knowledge required, namely, the knowledge element k(h,o), defined earlier as information about whether hypothesis h can cause observation o.

Case 1. Linear causal accounts.

Consider a causal simulation of the type represented in Figure 1. In the figure s1,.. sn, are unobservable internal states of the device, and each state gives rise to only one relevant successor causal state. Whether to store "$h \longrightarrow o$" depends on the value of n. The savings here are linear.

[Figure 1 about here]

Case 2. Multiple consequences.

Suppose each cycle of propagation produces more than one relevant successor state, e.g., "leaky valve $\longrightarrow$ reduced cooling water," and "leaky valve $\longrightarrow$ shorting of controller circuit." Figure 2 is an example of a causal representation of this type. In the figure some of the simulation paths converge on o, while others go to states which are not observationally significant. At run-time, that is, at the time of diagnostic problem solving, starting with h, the problem solver will have to do a search to determine if h causes o. Depending upon the how branching at one node is related to branching choices taken at other nodes, the search process may be combinatorial, but it will be at least polynomial. Compiling the knowledge "$h \longrightarrow o$" here can produce considerable saving during run-time in either case.

[Figure 2 about here]

For sufficiently complex systems, there is no guarantee that causal paths leading to the observation of interest, even if they exist, would be found within a fixed time. This is because a priori we don't know if additional propagation of consequences might result in observational states of interest.

The complexity of search in the SBF model space is especially high when the device has a number of complex subsystems at different levels of abstraction any one of which might be chosen for propagating effects. My favorite simple example is from the medical domain. One of the functions of the liver is to produce bile which is deposited in the duodenum where it is broken down as it is used in the digestive process. When there is obstruction in the bile duct, one cycle of simulation leads to "bile duct obstruction (h) $\longrightarrow$ no bile in the duodenum (o1) and bile retained in the liver (o2)." However, neither o1 nor o2 is easily observable. But, as it happens, due to complex interactions between liver and renal systems, "o2 $\longrightarrow$ direct bilirubin in urine (o3)," where o3 *is* observable. Even when SBF models for the various physiological subsystems are available, identifying causal paths such as "bile duct obstruction $\longrightarrow$ bilirubin in urine" is in the nature of a small medical discovery. Such discoveries are prize possessions of the diagnostic community. Any physician who does not have access to such

compiled diagnostic knowledge and who routinely attempts to perform simulations based on SBF models is unlikely to be very successful.

Case 3. Interaction of Causes.

A combination of malfunctions can cause a different set of observations than the union of their individual effects. This is in fact a fundamental source of the complexity of diagnostic reasoning for multiple faults. Since in theory we need to consider all combinations of malfunctions, the total amount of work in the simulation space can be substantial. In fact, most AI diagnostic systems make arbitrary cut-offs in the number of malfunctions that would be jointly considered, the single fault assumption being the most common.

If medicine is any guide, human problem solvers in fact have a rich pre-compiled store of lists of malfunction combinations which can potentially interact or be causally related. During diagnostic reasoning, the problem solver might first try to explain the data by only considering combinations in this list.

4.6 Combination of Models

There is a dual message that comes out of the analysis in the previous section. On one hand, diagnostic knowledge that helps map from observations to causes often requires extensive search in the space of causal states generated from SBF models, so caching those relations is a good idea. On the other hand, especially in cases 2 and 3 above, there can be no guarantee that any body of compiled diagnostic knowledge can be regarded as complete, except in the case of simple devices, due to the fact that the simulator does not know a priori whether or not additional propagation will result in an observation of interest. In fact, SBF models of devices are themselves unlikely to be complete. Before a structure/behavior model is constructed of a device, an a priori choice is made about the behaviors of interest that would be represented, and a particular diagnostic problem might have observations that require simulation of additional modalities of behavior not explicitly represented.

This situation is at the root of the empirical fact that almost all nontrivial human diagnostic reasoning uses a combination of access to compiled diagnostic knowledge and resort to a *variety* of SBF and other models. Diagnostic strategies usually combine compiled knowledge and SBF and other models in a number of ways. Here are some examples.

1. When knowledge for evaluating a hypothesis is missing, an appropriate version of an SBF model might be accessed selectively – selectively in the sense that only parts thought relevant to current hypothesis are invoked – to derive the needed knowledge. The MDX-2 system of Sticklen [4] uses functional models of physiological subsystems to compile elements of knowledge needed for specific diagnostic hypotheses.

2. A common diagnostic strategy starts by using reasonably complete diagnostic knowledge structures to perform diagnosis. If all the data are explained satisfactorily, the process stops. If not, SBF or other models are selectively accessed to look for causal pathways by which the current set of hypotheses can explain the unexplained data as well, or other hypotheses can be identified. A variant of this strategy occurs when the hypotheses explain all the data, but nevertheless it is thought that there may exist causal connections between the hypotheses that might result in a more satisfying and parsimonious diagnosis. SBF models, especially those that are richly indexed with information about function and behaviors at different levels of abstraction, can then be searched in a selective manner to try to establish causal connections. This is the strategy adopted by Punch [11].

3. Human diagnosticians often attempt to convince themselves that a diagnosis that they have arrived at makes sense by constructing causal stories to support the diagnosis. A knowledge system might first perform a diagnosis by means of a compiled knowledge structure, and then attempt to build a detailed causal explanation from the diagnoses to the observations by using SBF or other causal models. This is exactly the strategy adopted by Keuneke's research [12] on generating diagnostic explanations from functional representations of process systems. Note that the fact

that a causal path has been strongly hypothesized and its directions are also given by the diagnosis radically reduces the search in the simulation space.

5 Form and Content in Knowledge-Based Problem Solving

Does compilation change the content of a knowledge base? If one learns a fact that is simply a logical consequence of other facts, has there been a change in the content of one's knowledge? The distinction between form and content is a subject of on-going debate in AI, but the terms are themselves poorly defined or understood. In one sense of the word "content," a set of axioms has the same content as the set of all theorems that are derivable from it. But that is scant comfort for a problem solver who is trying to *prove* theorems in that axiom system. No inferential or problem solving activity *adds* content in this sense; in fact, throwing away details of the search and retaining only the successes *loses* content. But just as having an appropriate lemma might make proving theorems easier, directly having access to diagnostic knowledge will help in diagnostic problem solving. This is a form of operationalizing knowledge as has been discussed in research on Explanation-Based Learning.

In discussions on knowledge systems, two different senses of the word "form" are often conflated. One sense of it has to do with the vocabulary in which the knowledge needs to be expressed for a certain task. For example, we say that the classification method for the diagnostic task needs knowledge in a form that relates malfunctions to observations. This sense of form actually refers to the kinds of knowledge needed, that is, the kinds of things represented. For example, knowledge of could-be-caused-by relationships is needed for many methods for diagnosis. Form in this sense is thus, paradoxically, a statement about content of knowledge. The other sense of "form" has to do with the symbol-level representation adopted: whether the knowledge is represented in the form of rules, frames, sentences in predicate calculus, or whatever. I have argued for a number of years that there has been

entirely too much concern with form in the latter sense, and not enough on form in the former sense.

6 Concluding Remarks

There has been a certain confusion caused by the somewhat different senses of meaning different researchers have brought to terms such as "deep," "compilation," and so on. As one of the early participants in the discussion on this set of issues, I have attempted in this contribution to give the senses in which I have used the terms. The terms themselves are not in the long run as important as the distinctions that they are attempting to capture. The intuitions that underlie my work relate to the idea that there is a close relation between tasks, methods and types of knowledge needed for the methods. Knowledge needed for some types of tasks are themselves the result of problem solving using other types of knowledge. These intuitions have been the basis of the work by me and my colleagues on generic task architectures [13], and more recently, on task structures [14]. These intuitions also motivate my notions of depth and compilation.

In my view, the important research issues have to do with how a problem solver can flexibly move from one model to another in the pursuit of its goals. In Section 4.6 I discussed some examples of integration of different types of models for diagnosis. In [14] I discuss the role of a task structure in providing for a flexible integration of models.

Acknowledgements. I acknowledge support from DARPA and AFOSR, Contract F-49620-89-C-0110 and AFOSR grant 890250. I also owe thanks to Tom Bylander, Matt DeJongh, Tom Dietterich, Richard Fox, Ashok Goel, Yumi Iwasaki, John Josephson, Dale Moberg, Jack Smith, Jon Sticklen, and Mike Weintraub for useful comments on early drafts.

References

[1] Shortliffe, E. *Computer-Based Medical Consultations: Mycin*, Elsevier Scientific Publishing Co., 1976.

[2] Chandrasekaran, B., and Mittal, S. "Conceptual representation of medical knowledge by computer: MDX and related systems." *Advances*

in Computers, Academic Press, 1983, pp. 217-293.

[3] Clancey, W. J., "NEOMYCIN: Reconfiguring a rule-based system with application to teaching," in Clancey, W. J. and Shortliffe, E. (editors), *Readings in Medical Artificial Intelligence*, Chapter 15, pp. 361-381, Reading, MA: Addison-Wesley, 1984.

[4] Sticklen, J., MDX2: An Integrated Medical Diagnostic System, Ph. D thesis, Ohio State University, 1987.

[5] Newell, A. "The knowledge level," *Artificial Intelligence*, 18:87-127, 1982.

[6] Newell, A. and Simon, H. A. *Human Problem Solving*, Prentice-Hall, 1972.

[7] Chandrasekaran, B. and Mittal, S. "Deep versus compiled knowledge approaches to diagnostic problem solving," *Int. J. Man-Machine Studies*, pp. 425-436, 1983.

[8] Sembugamoorthy, V. and Chandrasekaran, B.. "Functional representation of devices and compilation of diagnostic problem solving systems," In Kolodner, J. L., and Riesbeck, C. K. (editors), *Experience, Memory and Reasoning*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1986.

[9] Bylander, T., "Some causal models are deeper than others," *Artificial Intelligence in Medicine*, vol. 2, 1990, pp. 123-128.

[10] Rosenbloom, P. S., Laird, J. E., and Newell, A. "SOAR: An architecture for general intelligence," *Artificial Intelligence*, 33:1-64, 1987.

[11] Punch, W. F., A Diagnosis System Using a Task-Integrated Problem Solver Architecture (TIPS), Including Causal Reasoning, Ph. D thesis, Ohio State University, 1989.

[12] Keuneke, A, Machine Understanding of Devices: Causal Explanation of Diagnostic Conclusions, Ph. D thesis, Ohio State University, 1989.

[13] Chandrasekaran, B "Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design," *IEEE Expert*, 1(3):23-30, 1986.

[14] Chandrasekaran, B. "Design problem solving: a task analysis," *AI Magazine*, vol. 11, no. 4, Winter 1990, pp. 59-71.

Figure 1. Case 1. Each state has a single causal consequence

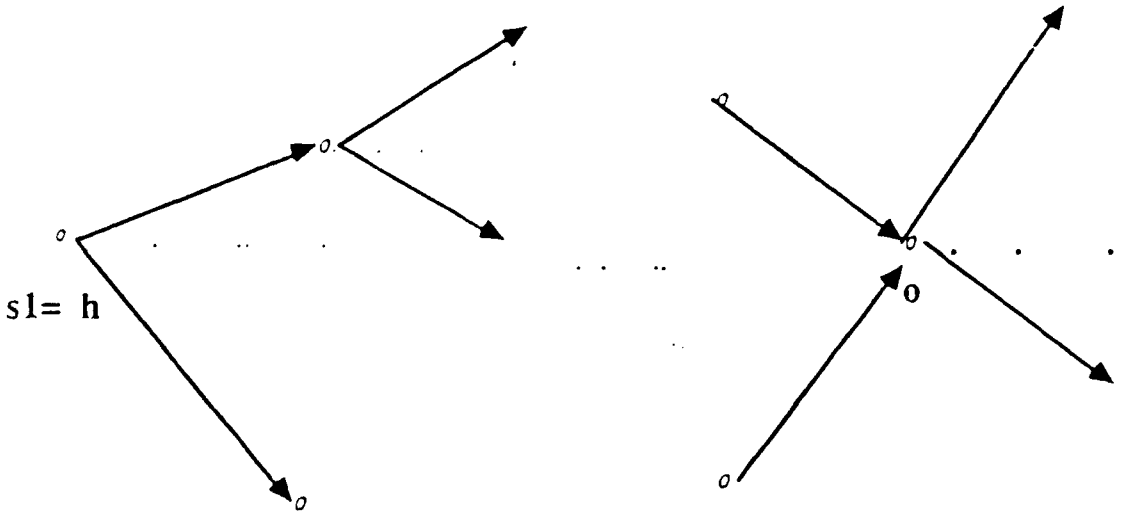


Figure 2. Case 2: Multiple causal consequences from each state

Reasoning Visually about Spatial Interactions

N. Hari Narayanan
Laboratory for Artificial Intelligence Research
Department of Computer and Information Science
Ohio State University
Columbus, OH 43210, USA
Tel: (614) 292-1413
email: narayan@cis.ohio-state.edu

B. Chandrasekaran
Knowledge Systems Laboratory
701 Welch Road, Bldg. C
Stanford University
Palo Alto, CA 94304, USA
Tel: (415) 723-1667
email: bxc@sumex-aim.stanford.edu

Abstract

In this paper we discuss how diagrams can be used for reasoning about spatial interactions of rigid objects. Our purpose is to provide a computational approach that emulates the human capability of predicting interactions of simple objects depicted on two dimensional diagrams. Three core aspects of our approach are a dual representation scheme that has symbolic and imaginal parts, the use of visual operations such as scanning on the imaginal part to extract spatial information, and visual cases that encode experiential knowledge and play a central role in the generation of spatial inferences. We first illustrate the approach with a detailed example. Next, we discuss related work which indicates that reasoning with images is an emerging and promising area of research. Finally, the significance of this work is discussed.

Major area: cognitive modeling. **Subarea**: qualitative spatial reasoning

Declaration: This paper is neither currently under review for a journal or another conference nor will it be submitted elsewhere during the review period of IJCAI.

1. Introduction

Humans quite often make use of spatial information implicit in diagrams to make inferences. For example, consider the cross-sectional of a simple pressure gauge that works by balancing air pressure against a spring-loaded piston inside a cylinder, as shown in figure 1. Reasoning about the functioning of this device from the diagram involves reasoning about spatial processes constrained by the cavity of the device. In this case particularly, it requires an ability to comprehend the constrained motion of the piston inside the cylinder. Similarly, anyone familiar with the operation of gears will be able to solve the problem posed in figure 2 by imagining the rotary motion of gear1 being transmitted to the rod through gear2, resulting in the horizontal translation of the rod until it hits the wall. In such situations humans reason about spatial interactions not only by using conceptual knowledge, but also by extracting constraints on such interactions from a perceived image. This integrated use of visual knowledge (about spatial configurations) from the diagram and conceptual knowledge (such as the rigidity or plasticity of objects involved) is a very interesting phenomenon. In this paper we illustrate a computational approach that emulates this capability for solving simple motion prediction problems.

2. The Approach

2.1 Motion Prediction Problems

The class of problems we address is the following. Given a two dimensional diagram of a rigid body configuration along with one or more initial motions of objects, predict the subsequent dynamics of the configuration. Figure 3 shows a prototypical example.

2.2 Cognitive Inspiration

Introspective reports of people, when given diagrams like figure 3 and asked to predict motions, indicated that by looking at the diagram they were able to visualize the motion of one object causing that of another through physical contact. They appeared to be using the image of the diagram in front of them directly to simulate motions in their minds. There is also considerable evidence in cognitive psychology for the use of mental images by people when solving spatial problems (Kosslyn, 1981). The reports exhibited the following characteristics.

(1) Given a diagram depicting the problem, humans quite rapidly focus on localities of potential interactions.

(2) People also seem to simulate or project the motion to determine the nature of interactions that will occur.

(3) For reasoning about the dynamics (e.g., how will motion be transmitted after a collision?) humans bring conceptual knowledge (e.g., gears are rigid objects) and experiential knowledge (e.g., if an object collides with another, it typically transmits motion in the same direction) to bear on the problem.

We have developed an approach that exhibits these capabilities.

2.3 Representation

A motion prediction problem is specified by a scene depicting the spatial configuration of the objects involved, and information about their properties. The scene itself is represented as a hierarchical multi-level description in which all but the lowest levels are symbolic and the lowest level is imaginal. Thus it is a dual representation. The resolution increases with levels, so the top level contains the

coarsest description. Each level of the representation contains descriptions of shapes of objects in the scene being represented and their relative spatial configurations using parametrized descriptors. At a level of low resolution an object may be represented by a single shape descriptor while at a level of high resolution it may be represented in terms of shapes of its delineative parts. The imaginal part of the representation is a 2-dimensional pixel array of fixed width and height in which a scene is depicted by the boundaries of objects in the scene. Thus the "image" is a boundary-based rendering and is implemented as a bitmap. An interesting property of this representation is that it directly captures, in its imaginal part, spatial information such as the restriction of motion freedom of an object due to obstacles. This representation has been inspired by Marr's work (Marr & Nishihara, 1978) and by a representation for mental images, called an image symbol structure, that we proposed previously (Chandrasekaran & Narayanan, 1990). Figure 4 shows a part of the representation of the motion prediction problem in figure 3.

2.4 Reasoning

Figure 5 shows a process description of our approach. One key component of this process is the application of visual methods to retrieve spatial information from the image and to modify it. Visual methods are composed from a set of basic visual operations such as scanning and boundary-following. Process steps such as detecting interesting regions, locating surfaces of potential interaction, and reasoning about the dynamics of object configurations are all done through visual methods operating on the dual representation. The following example will illustrate the role of visual methods. For more details on these, see (Narayanan, forthcoming). Applying

visual methods to determine the next deliberative state corresponds to "imagining" steps in introspective reports. Once a deliberative state is determined, however, experiential knowledge becomes relevant. A deliberative state is a decision making state in which deliberate reasoning (in case of humans) or knowledge-based reasoning (in case of computers) is employed to guide following steps by determining subsequent dynamics of the object configuration. A configuration in which two objects are colliding or one in which a previous contact between two objects has been removed due to motion are examples of deliberative states.

Experiential knowledge has a central role in deciding how to proceed from a deliberative state. We believe that the knowledge humans use in such situations is mostly acquired through experience. Experiential memory is considered to be an organized and indexed collection of cases (Schank, 1982). Therefore, representational structures called "visual cases" have been developed to encode experiential knowledge in the computer. Each case represents a typical spatial event. These are called "visual" because the configurational information encoded in a case is imaginal in nature and this information is the "key" by which relevant cases get selected during reasoning. Since cases are acquired from experience, they may not be logically parsimonious or mutually exclusive. A visual case has three parts. One is the visual information. The second is non-visual knowledge that qualifies the visual part further and it is used for deciding the applicability of a case to a particular situation. The third part is a predicted event affecting objects in the spatial configuration represented by the case. This event may specify a state change (e.g., a directional force being applied on an object), a continuous change (e.g., an object

moving in a particular direction), etc. The intent of visual cases is to encode simple chunks of experiential knowledge about typical spatial events that humans have. An example of this is "a rigid object resting on a rigid flat surface, when collided by a moving rigid object, will tend to slide in the same direction". A corresponding visual case is shown in figure 6.

The knowledge-based reasoning step in the process description thus involves the selection and application of visual cases to predict events that follow a deliberative state. After a deliberative state is computed, the next action is retrieving relevant cases using the visual parts of cases as indices. From among these, applicable cases are selected by using conceptual knowledge to verify the non-visual parts of the cases. Events predicted by the applicable cases are further pruned by using current configurational information from the image. The remaining events serve to guide subsequent steps of reasoning. Thus a visual case application results in a spatial inference being made, based on the current configuration as depicted on the image as well as other knowledge. A visual case brings conceptual knowledge to bear on spatial reasoning and is a computational analog of Pylyshyn's cognitive penetrability argument (Pylyshyn, 1981) concerning the influence of tacit knowledge on mental imagery.

Now we illustrate this process in operation for the problem in figure 3. Reasoning proceeds by generating goals and subgoals, selecting methods, and applying them in accordance with the process description. Figures 7 - 11 illustrate various stages and results of reasoning. Each figure is a snapshot that shows a

partial execution trace in terms of goals/subgoals generated and methods applied, with the last applied method appearing in boldface, and its effect on the bitmap. Figure 11 depicts a stage at which one iteration of the process is complete. After this execution returns to the step of locating surfaces of potential interaction. This process continues until no more changes occur in the configuration. The cumulative result is given in figure 12 which shows the final configuration of objects.

3. Related Work

Research on mental imagery (Shepard & Cooper, 1982) provides support to our belief that reasoning with imaginal as well as symbolic representations can be a promising paradigm for intelligent systems. Despite intuitively compelling evidence for the use of imagery by humans, there has not been much work in artificial intelligence toward endowing machines with a similar capability. Existing work on kinematics, for example, (Faltings, 1987; Nielsen, 1988) takes the approach of using algebraic descriptions and configuration spaces for solving problems like reasoning about the motion of interlocked gear pairs. However, the visual approach is applicable to the same class of problems (Narayanan & Chandrasekaran, 1990) and can provide approximate solutions quickly. An early program that utilized diagrams was WHISPER (Funt, 1977) which addressed rotation, sliding, and stability of blocks-world structures. More recently, work on using pictorial or "analogical" representations for simulating the behavior of strings and liquids in space has been reported (Gardin & Meltzer, 1989). Luc Steels (1988) has also suggested the use of analogical representations for reasoning about the physical world. The promise of pictorial representations is evident in Shrager's (1990) work

on commonsense perception as well. He casts theory formation as an act of reinterpretation of knowledge grounded in imagery and describes a computational model of theorizing about lasers in which propositional descriptions are grounded in an iconic memory. The instantiation of process views and the resulting mental imagery-like animation in this laser model is similar to the application of visual cases.

4. Concluding Remarks

We have illustrated a novel approach to reasoning about spatial interactions. The advantages of using diagrams in this approach arise from the property that spatial information such as obstacles to motion or pathways that guide motion are directly evident in images. Therefore this approach can provide approximate solutions quickly. Our approach is not only intuitive, but flexible as well. Objects which have irregular shapes that will make their algebraic representations complex can be represented and reasoned about in the same way as regular objects if diagrams are used. Reasoning visually can provide initial approximate solutions that can guide the application of more quantitative methods to spatially localized regions where more precision is required. As Forbus and colleagues rightly point out (Forbus, Nielsen & Faltings, 1987), there can be no purely qualitative method for spatial reasoning. What is required is to develop qualitative and quantitative methods and integrate them so that qualitative ones provide approximate solutions that serve to focus the application of computation-intensive quantitative methods to only those aspects of the initial solutions that require more precision or further refinement. We believe that systems capable of reasoning about motion transferring devices such as gear trains

and acquiring models of simple mechanical devices from their cross-sectional diagrams can be developed using the visual approach. This has great significance to engineering applications of artificial intelligence in terms of diagnostic and design systems capable of reasoning with diagrammatic representations of devices.

Acknowledgments: A discussion with Stephen Kosslyn provided inspiration during the early stages of this work. Jeff Shrager helped clarify many issues. This research is supported by DARPA & AFOSR contract # F-49620-89-C-0110 and by AFOSR grant # 890250.

References

Chandrasekaran, B., & Narayanan, N. H. (1990). Integrating imagery and visual representations. *Proc. 12th Annual Conference of the Cognitive Science Society*, Boston, MA, 670-678.

Faltings, B. (1987). Qualitative kinematics in mechanisms. *Proceedings IJCAI-10*, Milano, Italy, 436-442.

Forbus, K. D., Nielsen, P., & Faltings, B. (1987). Qualitative kinematics: a framework. *Proceedings IJCAI-10*, Milano, Italy, 430-436.

Funt, B. V. (1977). WHISPER: a problem solving system utilizing diagrams and a parallel processing retina. *Proceedings IJCAI-5*, Cambridge, MA, 459-464.

Gardin, F., & Meltzer, B. (1989). Analogical representations of naive physics. *AI Journal*, 38, 139-159.

Kosslyn, S. M. (1981). The medium and the message in mental imagery: a theory. *Psychological Review*, 88, 46-66.

Marr, D., & Nishihara, H. K. (1978). Representation and recognition of the spatial organization of three dimensional shapes. *Proceedings of the Royal Society*, 200, 269-294.

Narayanan, N. H. (forthcoming). Visual reasoning: a novel paradigm for intelligent systems. Doctoral Dissertation, Department of Computer & Information Science, Ohio State University, Columbus, Ohio.

Narayanan, N. H., & Chandrasekaran, B. (1990). A visual approach to qualitative kinematics. *Proc. AAAI-90 Workshop on Qualitative Vison*, Boston, MA, 72-76.

Nielsen, P. (1988). A qualitative approach to mechanical constraint. *Proc. AAAI-88*, St. Paul, MN, 270-274.

Pylyshyn, Z. W. (1981). The imagery debate: analogue media versus tacit knowledge. *Psychological Review*, 88, 16-45.

Schank, R. (1982). *Dynamic memory: a theory of learning in computers and people*. New York: Cambridge University Press.

Shepard, R. N., & Cooper, L. A. (1982). *Mental images and their transformations*. Cambridge, MA: MIT Press.

Shrager, J. (1990). Commonsense perception and the psychology of theory formation. J. Shrager & P. Langely, (Eds.), *Computational models of scientific discovery and theory formation*. San Mateo, CA: Morgan Kaufmann.

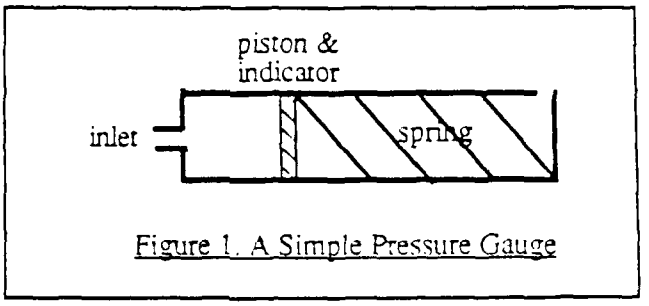


Figure 1. A Simple Pressure Gauge

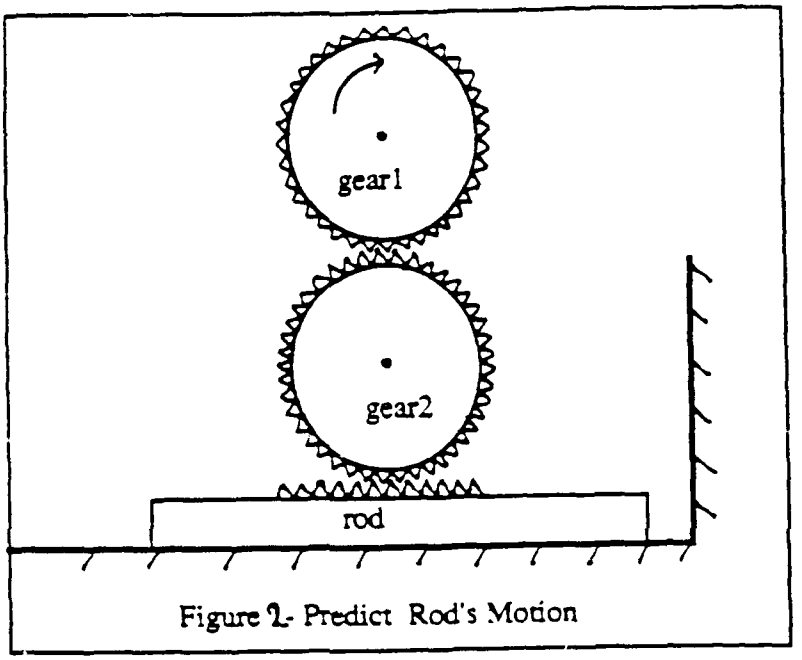
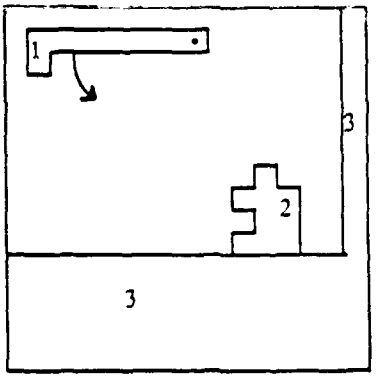


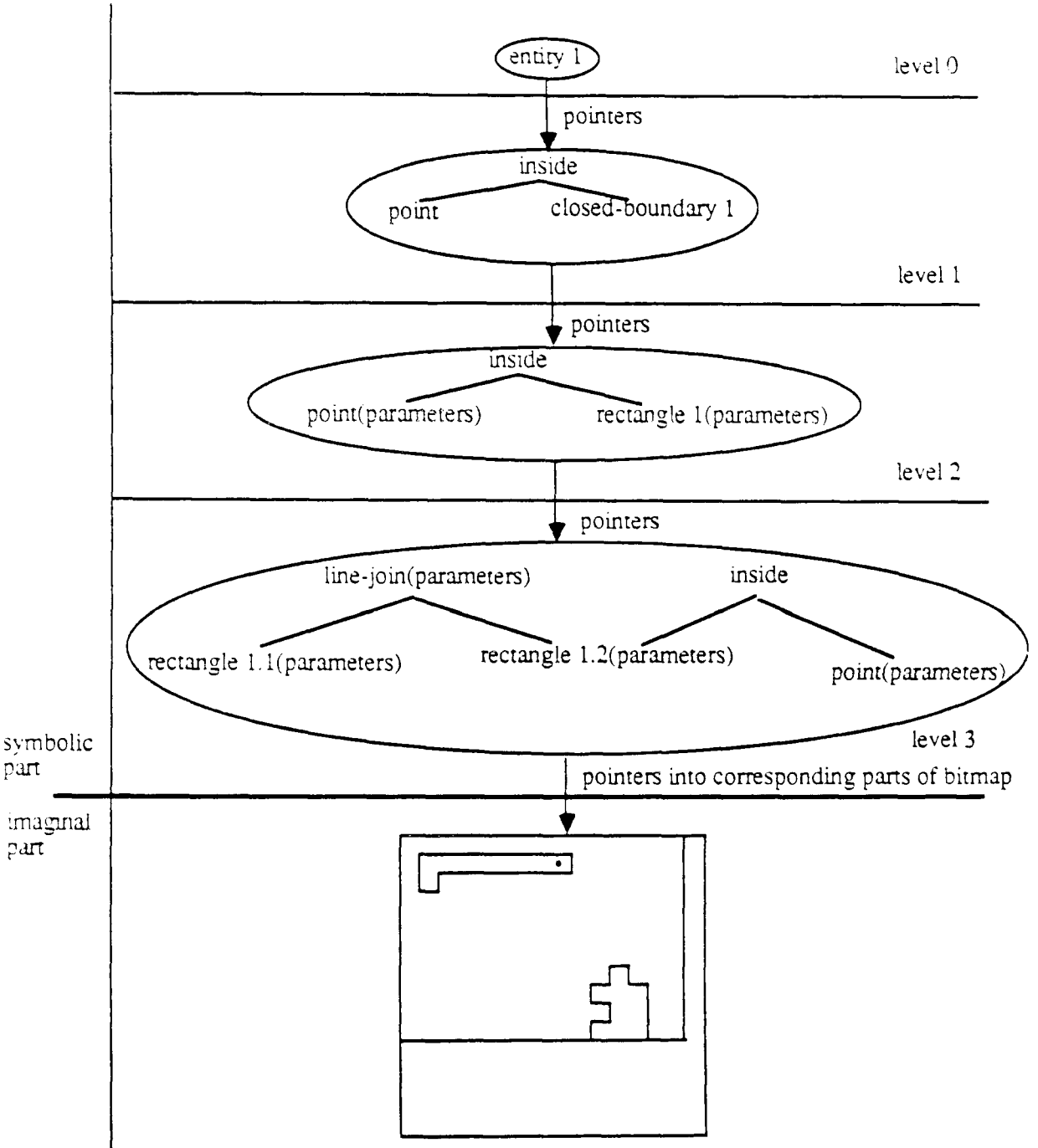
Figure 2- Predict Rod's Motion



1 : Rigid hinged rod
2 : Rigid block
3 : Rigid surfaces
2 can slide on 3

Figure 3. A Motion Prediction Problem

Figure 4. A portion of the dual representation



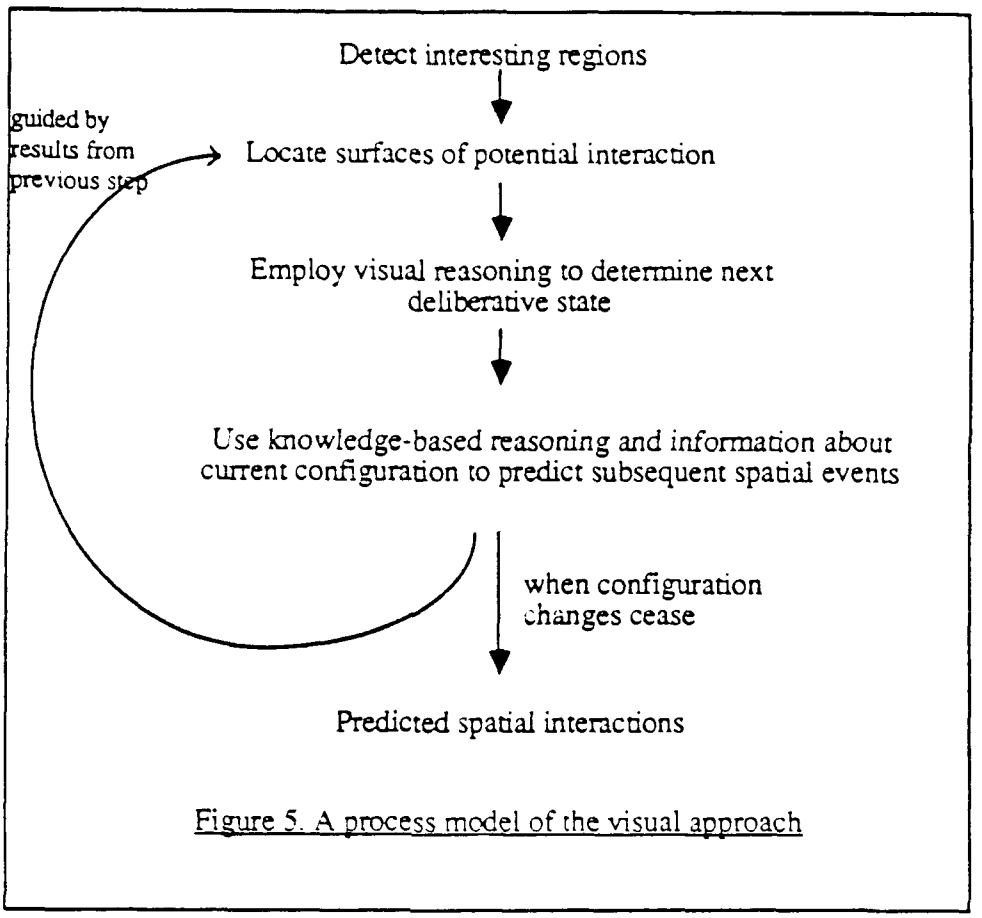


Figure 5. A process model of the visual approach

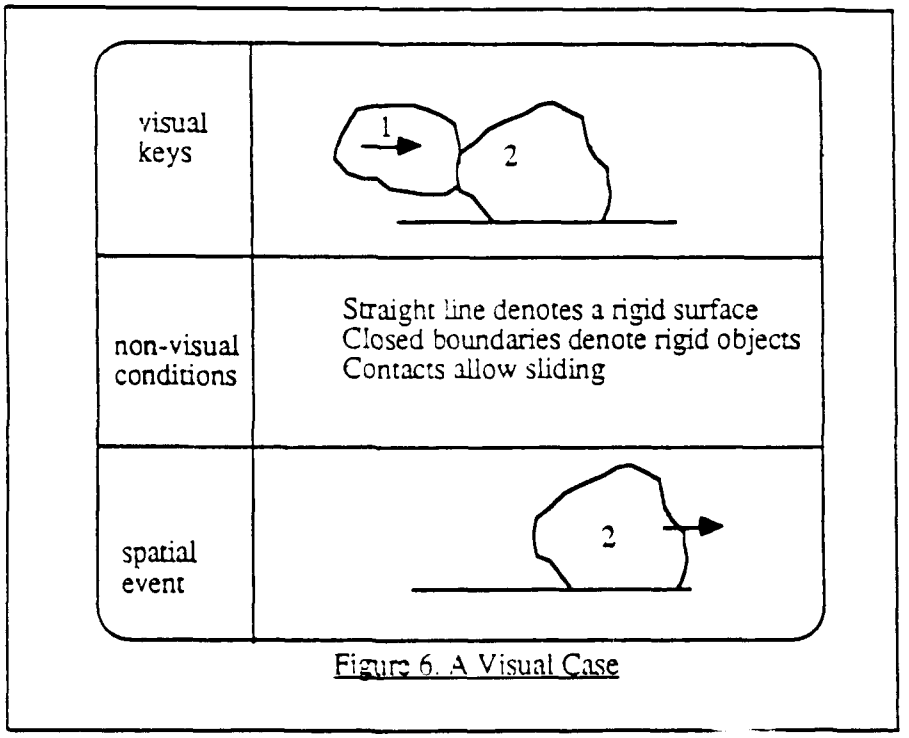


Figure 6. A Visual Case

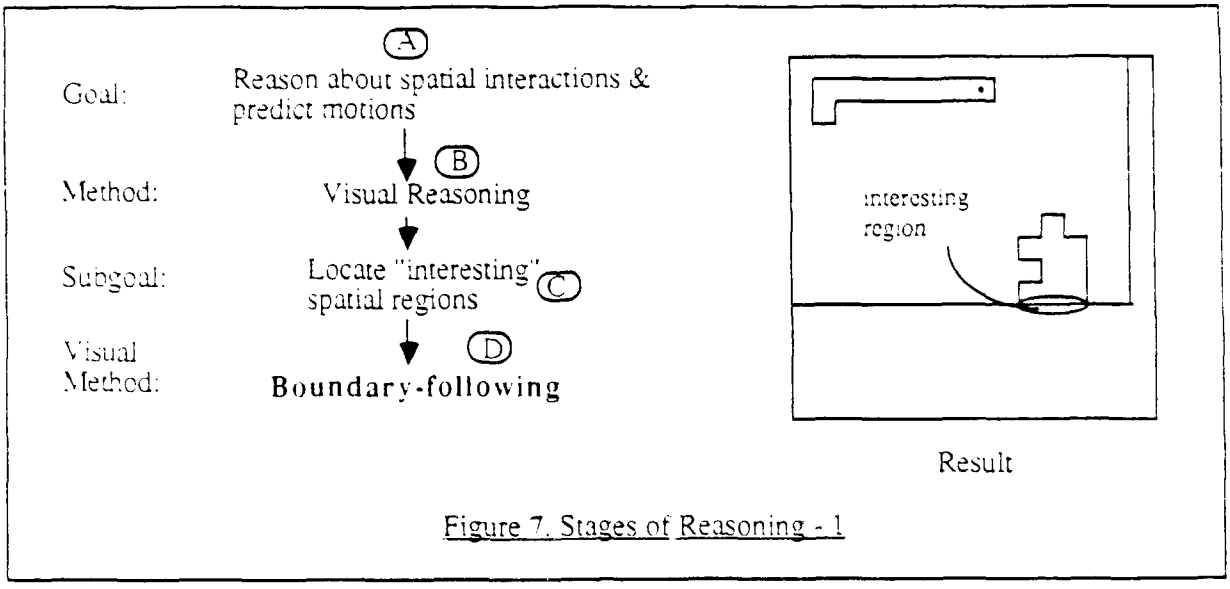


Figure 7. Stages of Reasoning - 1

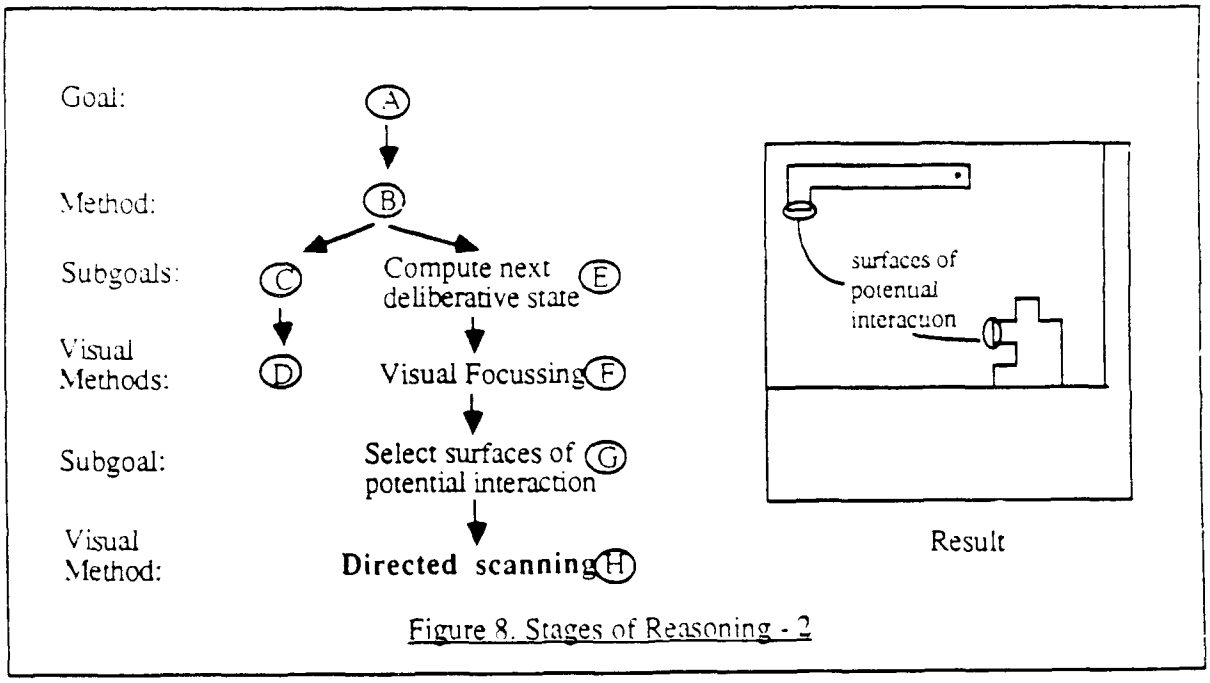


Figure 8. Stages of Reasoning - 2

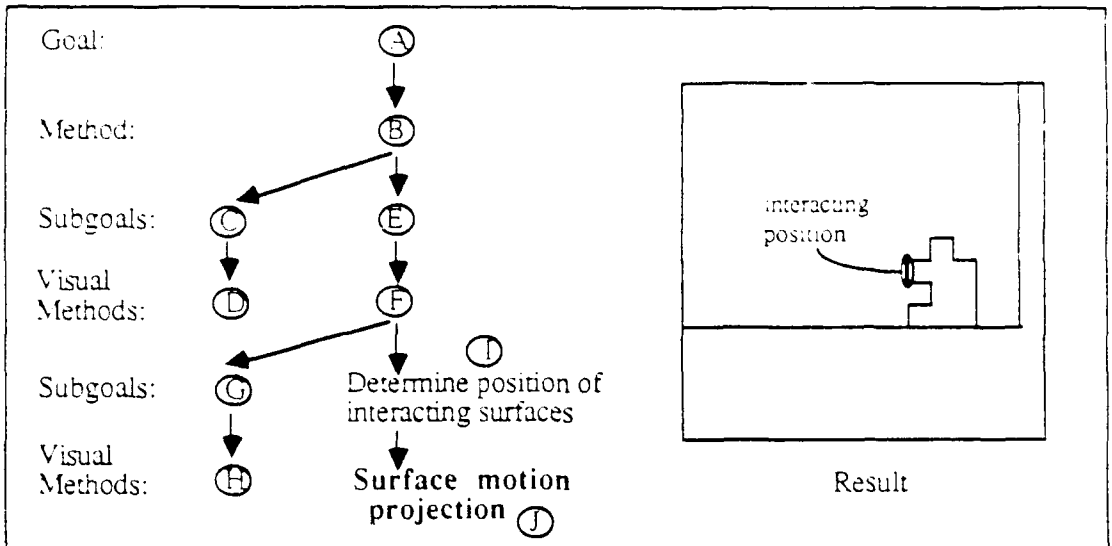


Figure 9. Stages of Reasoning - 3

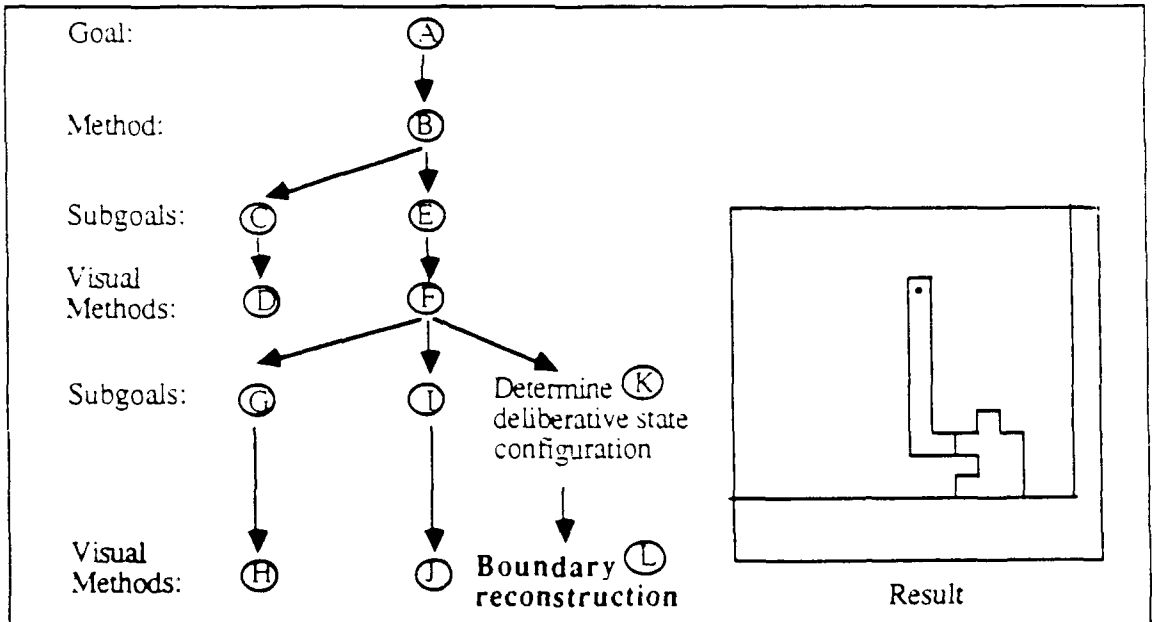


Figure 10. Stages of Reasoning - 4

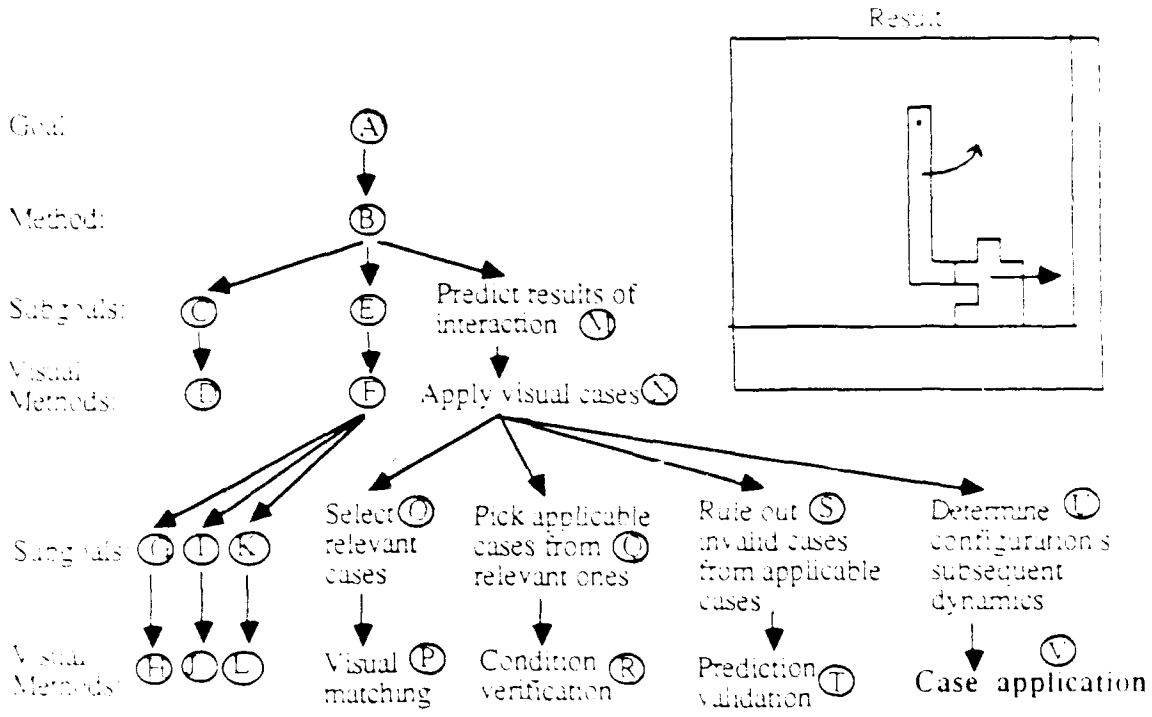


Figure 11. Stages of Reasoning - 5

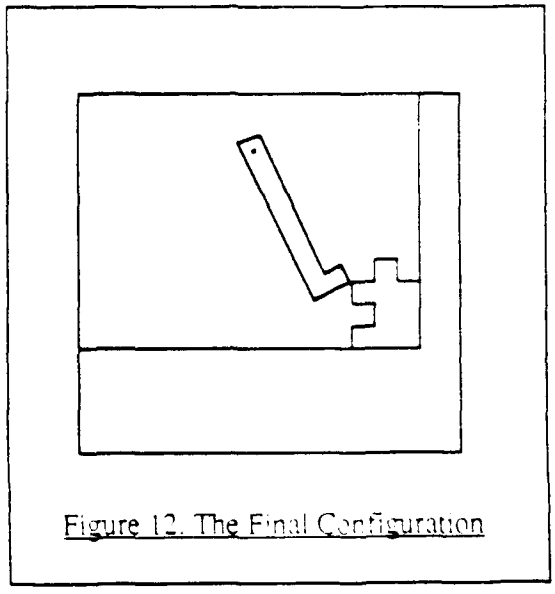


Figure 12. The Final Configuration

Response to "Is Human Cognition Adaptive?" by J. R. Anderson

B. Chandrasekaran
Laboratory for AI Research
The Ohio State University
217 Bolz Hall
Columbus, OH 43210

DRAFT February 27, 1991

Evolution is more likely a "satisficer" than an optimizer. With that proviso, it does appear from Anderson's paper that a surprising amount of the details of human cognitive behavior can be explained as a satisficing, if not an optimal, response to the structure of the environment. But what consequences follow for various research programs is not quite clear. I'm an AI person, not a traditional cognitive psychologist. In addition to whatever explanatory powers psychologists want from their theories, I want the theories to have *design-prescriptive* powers as well, that is, I want theories to tell me how to *create* mind-like entities. Historically, the route for this sort of progress has come from mechanistic explanation (ME) accounts of mental phenomena. Thus I read the target article from the perspective of what rational analysis (RA) has to say about ME.

Anderson's views on this range from his belief, in "overenthusiatic moments," that RA can supplant ME, to a more sober suggestion that RA accounts place constraints on ME accounts. Finally, he actually displays an ME account for categorization which actually *implements* his RA in the sense that it can be thought of as literally estimating the various probabilities involved.

Given a cognitive agent in an environment engaging in a certain behavior, how to allocate an explanation of the behavior between the structure of the agent and the structure of the environment is an issue that has been discussed before in psychology and AI. For example, consider Simon's ant (Simon, 1982): It produces a path of great complexity on the beach, but a great deal of this complexity is explained by the properties of the environment, that is, the shape of the sandhills on the beach. In this case, a roboticist charged with producing an artificial ant would be making a mistake if he thought that the ant had an internal structure that somehow had an encoding of the path. He would not only be wrong in allocation of

explanations, but he would be constructing an ant that doesn't work right.

But the sort of problems that Anderson is concerned with are not of this type. Here the explanations are not necessarily allocated between the internal structure of the agent and the structure of the external environment; *both* simultaneously account for the behavior, albeit in different ways. We need to define some notations in order to help with the discussion of this idea.

Let E stand for the environment and Si and bi for structure of an agent and its behavior at time i. Let M stand for any body of mechanisms within the agent which takes as input Si, bi and the response of the environment and produces as output S(i+1), that is, it is some sort of learning or structure-modifying function. Let us assume for the purposes of this discussion that the structure of E is invariant in time, and that we are interested in the steady state properties of S and b, that is, for i = infinity.

M itself may be a complex collection of mechanisms with different time constants: one in the scale of biological evolution, perhaps another one in the scale of cultural evolution and finally yet another one in the scale of learning by an individual.

For the question "Why is b the way it is?" we have two types of answers. One is that b is the way it is because S of such-and-such type produces it (traditional ME). The other is that b is the way it is because it is optimal for E (RA), but this story has a sub-plot: M modified S such that S was optimal for E, i.e., it could produce an optimal b. Both answers involve S, sooner or later.

If our aim is to make agents which display behavior b, we either need to know Sinf, or we need to know M, Sinit and have enough time in which to let M shape the S into Sinf. For the latter alternative, depending upon whether Sinit reflects the situation at the beginning of the individual, the culture, or some point in biological evolution, we are talking of a more or less practical program.

Of course, in the above, I have accepted the RA hypothesis that b is optimal. But, as Anderson acknowledges in the concluding section of the target article, some b's may not be optimal after all. It seems to me that whether b is optimal depends on the following things:

i. The presumed goal of the agent. If a behavior b is not optimal for goal g, perhaps it is optimal for goal g'. In some sense we can go shopping for goals. (Anderson is admirably careful about this issue in the examples that he has studied, that is, his statement of goals does not seem problematic, but it is not clear how long this fortunate state of affairs will last. For example, Marr assumes that a goal of the human visual system is to produce an account of 3-d shapes of the objects in the scene, but why is that not a reasonable goal for the frog's visual system as well? In general why wouldn't't any goals that we would ascribe to the human visual system not be appropriate goals for the frog's as well? In order to get RA off the ground, we will have to make additional assumptions, some of them about the

structure of the respective visual systems.)

ii. The properties of Sinit and M relative to the search space in which the specifications for optimal Sinf lie. Perhaps S will never get to the optimal Sinf. Thus for a specific cognitive function, we will not know until after RA and data analysis are complete, whether in fact b is optimal. In this sense, as a general research program, RA is asserting that b is optimal whenever it is.

My points in the above have been that ME and RA are complementary analyses, not alternatives, and that the RA program is not unambiguous in its methodology. Now I want to examine the claim that RA places strong constraints on ME.

Optimality of an agent's behavior does not imply that the agent is using explicit optimization to produce the behavior. This has been a pet peeve of mine about quite a bit of work in AI which assumes (i) that the job of an intelligence is to produce correct or optimal answers, and (ii) the mechanisms for production of intelligent behavior should implement normative methods of producing optimal answers, most commonly some form of logic or Bayesian analysis. On the contrary, it seems to me that it is neither necessary nor desirable that the mechanisms of behavior production are explicit implementations of normative behavior.

I need to clarify some terms before I proceed. The word "structure" of cognition is a bit too vague. We can assume that what is meant by that word is, in information processing language, two things: a mechanism and some content that has been put into the mechanism. To use some concrete examples, one proposal for a cognitive mechanism is a search engine, the latest example of the proposal being the SOAR architecture of Rosenbloom, et al (Rosenbloom, Laird and Newell, 1987). Such a mechanism corresponds to a language in which specific programs with specific content can be written. Thus Soar can be programmed to have knowledge about some domain and methods. A Soar machine, so programmed, can actually work on problems in that domain. The metaphor of a programming language does not restrict the above idea to symbolic mechanisms. The PDP style connectionist research similarly specifies an abstract mechanism, but that mechanism itself needs to be given content to solve specific problems. For example, when one designs a PDP style network to solve word recognition, one has in fact used the abstract "programming language" of PDP style connectionist mechanism to produce a specific "program" of that type.

In my view what is interesting about both these (and many other) mechanisms that have been proposed for cognition is that they can be used to implement both optimal and non-optimal methods for specific goals. In fact this property seems to be especially desirable, since the agent can adapt itself to changes in the environment without changing the basic mechanism, because it is neutral with respect to optimal and non-optimal algorithms. I want to give two examples of this.

I am told that frogs' visual systems are so organized that on danger they jump towards color blue and away from color green. It has been proposed that this is in fact optimal behavior: blue represents a body of water and safer for the frog, and green represents

land, full of predators. The neural mechanisms that implement this strategy could just as easily implement some other strategy of color preference. The RA that suggests that the optimal behavior is "jump to blue" is putting no constraints whatsoever on the basic neural mechanisms. Of course, such an RA *is* placing a constraint on the *content* of the mechanism, namely, that it should be programmed to prefer blue to green.

Similarly, I am told that during the plague in medieval Europe, some villages, on hearing of the breakout of the disease in a nearby village, engaged in a ritual of dancing at night near the village dump, making loud noises with pots and pans. As it happened, such villages had a smaller chance of catching the infection. A modern-day RA would show that this was actually optimal behavior, since the ritual kept the rats away from the dumps and consequently from the village. Many cognitive mechanisms can implement this strategy, including the following pair: a more or less random behavior generating mechanism that first conceived of some version of the ritual, and a cognitive mechanism that in some way remembered and passed on the ritual. Villages that survived were more likely to pass the ritual on. The same mechanisms could be used to implement relatively ineffective strategies.

Depending upon what is meant by the word "structure," RA can be thought of as giving clues about the structure of cognition. If by structure is meant abstract mechanisms, in general the case is less compelling. If by structure is meant the totality of abstract mechanism plus content, it seems quite reasonable to say that RA can give clues about structure, since, as seen in the above examples, it gives clues about the content.

This brings me to the relation of Anderson-style RA to Marr's approach. Anderson proposes that RA is similar to Marr's computational level. It is true that Marr, in his work on vision, proposes that we should start by asking "What are the goals of computation?" and "What is available in the image?," the latter question, in its generalization, being construed as "What is the nature of the environment?". However, while Marr uses his analysis of what is in an image to constrain what he could plausibly expect the visual system to be computing, his computational level account of vision is really a proposal about the *structure* of the visual system. Marr did not infer the existence of a level called "2 1/2-d sketch" purely from the properties of the image and the hypothesized goal of the visual system. It was a *hypothesis* by him, inspired by the analysis of the image no doubt, but nevertheless a hypothesis. The computational level, to use Newell's term, avoids symbol-level commitment to how the computation is implemented, but remains nevertheless a partial specification of a structure. The spirit behind Marr's levels is close to that behind Newell's distinctions between knowledge and symbol levels (Newell, 1982) in that both are attempts to develop a way of talking about structure without being tied down to the incidental aspects of implementation, but neither is an attempt to avoid specifying the structure needed for explanation.

I have tried to clarify the relationship of Marr's computational level and Anderson's RA, because I think the former still hews to the ME program. I can actually try to implement Marr's 3 stages (using additional commitments of course) to make a vision machine. I can't in general implement an RA to make the corresponding cognitive machine. Anderson's classification net is not really a counterexample, since as I have argued, in general we do not

want to be committed to literal implementations of optimizing methods to achieve optimizing behavior.

In summary, I have supported that side of Anderson that believes that RA and ME are complementary. I have also argued that in general RA may give guidance, not about the abstract mechanisms of cognition, but about their content.

Before concluding, I'd like to express my admiration of Anderson's piece as a tour de force of analysis and writing that illuminates the relation between behavior, structure of the agent and the environment. I think RA also helps provide arguments for why AI should worry about natural (i.e., human) intelligence. Often AI people make a fairly strong distinction between human and machine intelligence, and claim that there is no reason to base our mechanically intelligent agents on the structure of human cognition. If we want our machines to share our goals and operate intelligently in the sort of environments we operate in, we had better look to the structure of human intelligence for inspiration, since according to RA, it is probably pretty optimal for the task.

Acknowledgment: Support provided by AFOSR Grant 89-0250 in the preparation of this note is gratefully acknowledged.

References:

Newell, A. (1982). The knowledge level. *Artificial Intelligence*, 18, 87-127.

Rosenbloom, P. S., Laird, J. E., and Newell, A. (1987). SOAR: An architecture for general intelligence. *Artificial Intelligence*, 33, 1-64.

Simon, H. A., (1982). *The Sciences of the Artificial*, Second Edition. Cambridge, MA: The MIT Press.